VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Automatická správa požadavků v softwarových firmách

Automation Requirement Management in Software Companies

2014

Martin Chapčák

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Martin Chapčák**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Automatická správa požadavků v softwarových firmách
Automation Requirement Management in Software Companies

Zásady pro vypracování:

Cílem této práce je implementovat automatizovaný nástroj pro správu požadavků a financí klientů, kteří požadují specifickou úpravu stávajícího softwarového řešení. Tento nástroj bude automatický spravovat (pomocí API project management systému Redmine) dostupné lidské zdroje softwarové společnosti na základě dat dostupných z API Redmine. Systém bude rovněž sdružovat klienty se stejnými nebo podobnými požadavky.

1. Automatické rozpoznání a analýza procesů z logů opensource project management systému Redmine.
2. Odhalení a predikce možných budoucích problémů z dostupných informací z API Redmine.
3. Porovnání vhodnosti použití databáze typu SQL nebo NoSQL, s následným využitím nejvhodnější databáze zvoleného typu při implementaci systému.
3. Implementace systému pro správu klientských požadavků, jež bude na základě údajů automaticky obstarávat řízení change managementu.
4. Nasazení systému do realného provozu a vyhodnocení jeho funkčnosti.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

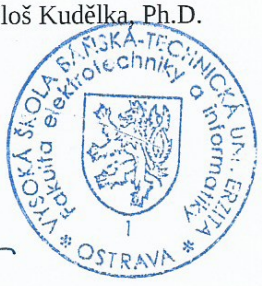
Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Michal Unzeitig**

Konzultant diplomové práce: Mgr. Miloš Kudělka, Ph.D.

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7.5 2014 .....

Souhlasím se zveřejněním této bakalářské/diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských/magisterských programech VŠB-TU Ostrava.

Text práce může být zveřejněn v plném rozsahu. Nebudou zveřejněny zdrojové kódy praktické části práce.

V Ostravě 7.5 2014

.....

Michal Unzeitig
jednatel ARTIO s.r.o.

Rád bych na tomto místě poděkoval pánům Ing. Michalu Unzeitigovi a Mgr. Miloši Kudělkovi, Ph.D., jejichž zkušené rady a připomínky mi s prací velice pomohly.

Abstrakt

Tato diplomová práce předkládá řešení pro správu požadavků klientů, kteří požadují zakázkovou úpravu již zakoupeného softwarového řešení. Společnost ARTIO s.r.o používá pro správu úkolů projektový řídící systém Redmine, který je pro správu klientských požadavků nedostatečný. Celý proces vyžaduje mnoho ruční práce a dlouhodobě způsobuje společnosti finanční ztrátu. Práce analyzuje požadavky a popisuje návrh systému, který zautomatizuje a zefektivní proces přijímání a zpracování požadavků. Implementované řešení navíc umožňuje finanční spoluúčast několika klientů na požadavku, což zvýší zisky společnosti a spokojenost klientů. Stávající systém Redmine navíc neobsahuje téměř žádné analytické nástroje, které by informovaly management o stavu projektů. Proto je implementován analytický nástroj, který analyzuje data ze systému Redmine, zobrazí je v přehledných grafech a identifikuje úkoly, které budou nebo mohou být opožděny. Tyto opožděné úkoly způsobují finanční ztrátu společnosti a proto je nutné je identifikovat co nejdříve.

Klíčová slova: Redmine, API, NoSQL, diplomová práce

Abstract

This master thesis shows solution for managing client's requests which want to change already bought software. Company ARTIO s.r.o uses project management system Redmine, which is unusable for complex management of requests. Whole process of contract requires a lots of manual work and costs lots of money in long-term use. This work analyze requirements and describe design of system which will make process of client's requirements automated and more effective. Implemented system provides solution for financial contribution of many clients on one request. This increases profit of company and also client's satisfaction. Redmine does not has any analytical tools informing management about state of projects. This is the reason why analytical tool which shows data in charts, and identifies tasks probably delayed in future is implemented. Delayed tasks must be identified as soon as possible because it cause financial loss.

Keywords: Redmine, API, NoSQL, diplomová práce

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CMS	–	Content Management System
CRM	–	Customer Relationship Management
HTML	–	Hyper Text Markup Language
MVC	–	Model-view-controller
ORM	–	Object-relational mapping
PHP	–	Hypertext Preprocessor
REST	–	Representational State Transfer
SOAP	–	Simple Object Access Protocol
SQL	–	Structured Query Language
SVN	–	Subversion
UML	–	Unified Modeling Language
URL	–	Uniform Resource Locator

Obsah

1 Úvod **5**
1.1 ARTIO 5

2 Joomla **6**
2.1 Extenze 6
2.2 Verze 7
2.3 ARTIO komponenty pro Joomla 8

3 Současné řešení **9**
3.1 Workflow 10
3.2 Redmine 12

4 Databáze **13**
4.1 RDBMS 13
4.2 SQL 14
4.3 NoSQL 16
4.4 SQL nebo NoSQL 18

5 Vize nového řešení **19**

6 Redmine API **20**
6.1 REST 20
6.2 Redmine API implementace 21

7 Asupport komponenta **24**
7.1 Požadavky 24
7.2 Podobná již existující řešení 28
7.3 Analýza 29
7.4 Návrh řešení 30
7.5 Implementace 35

8 Analytický systém **39**
8.1 Požadavky 39
8.2 Podobná již existující řešení 42
8.3 Analýza 44
8.4 Návrh řešení 46
8.5 Implementace 47

9 Vyhodnocení **50**
9.1 Poznámky k řešení 50
9.2 Vyhodnocení 50

10 Závěr **53**
10.1 Budoucí práce 53

11 Reference **54**

Seznam obrázků

1 BPMN diagram . . . 11
2 Diagram případů užití . . . 28
3 Diagram aktivit - Vložení nové podpory . . . 29
4 Sekvenční diagram - Vložení nového ticketu . . . 30
5 API knihovna . . . 32
6 Specializace pro CRM API . . . 33
7 Kontrolér . . . 33
8 Model . . . 33
9 Zobrazení . . . 34
10 Active resource . . . 34
11 Plugin . . . 34
12 Stavový diagram - stav veřejného (sdíleného) požadavku . . . 35
13 Diagram nasazení . . . 36
14 Diagram aktivit - nákup kreditů . . . 38
15 Diagram případů užití . . . 42
16 Diagram aktivit - Načtení dat . . . 44
17 Diagram aktivit - Zobrazení Gantt diagramu . . . 45
18 Třídní diagram . . . 46
19 Diagram nasazení . . . 47
20 Diagram aktivit neuronové sítě . . . 49

Seznam výpisů zdrojového kódu

1 Ukázkový SQL dotaz 1 15
2 Ukázkový SQL dotaz 2 15
3 Ukázkový SQL dotaz 2 15
4 Ukázka SOAP požadavku 21
5 Ukázka REST požadavku 21
6 Ukázka ActiveResource 21
7 Ukázka Redmine API 22
8 Vytvoření nového ticketu (issue) 37

1 Úvod

Společnost ARTIO s.r.o používá pro přidělování a spravování úkolů projektový řídící systém Redmine. Komunikace s klienty probíhá téměř výhradně pomocí emailu nebo systému Redmine, který s dodatečným modulem umožňuje odesílání emailu klientům. Ačkoliv je komunikace s klienty částečně automatizovaná, stále zbývá mnoho částí, které vyžadují manuální správu. Stávající řešení neobsahuje žádný nástroj, který by umožnil participaci více klientů na jednom požadavku. Systém Redmine rovněž neobsahuje vhodné analytické nástroje, které by informovaly management a celkovém stavu projektů. Vzhledem k zaměření společnosti (upřesněno v podkapitole ARTIO komponenty pro Joomla2.3) řeší společnost několik desítek požadavků a dotazů denně. Odborné požadavky musí být řešeny programátory, kteří pak nemají dostatek času pro vývoj. Tento stav je dlouhodobě neudržitelný, neboť jsou vývojáři zahlceni uživatelskou podporou a jejich drahý čas není věnován zvyšování kvality softwaru.

Současný stav je detailněji popsán v kapitole Současné řešení3. Rozdíly mezi relační a NoSQL databází jsou popsány v kapitole Databáze4. Vize popisovaná v kapitole Vize nového řešení5, slouží jako podklad pro implementaci, která je rozdělena na Asupport komponentu7 a Analytický systém8. Asupport komponenta řeší efektivnější komunikaci s klienty. Cílem je minimalizovat dotazy, kterými by se museli zabývat vývojáři a maximalizovat požadavky na individuální úpravy komponenty, které jsou placené klientem. Analytický systém poskytuje analýzu na základě dat z interního systému Redmine. Tento nástroj hraje klíčovou roli ve strategickém rozhodování managementu. Společným prvkem obou aplikací je napojení na systém Redmine, jež je popsáno kapitole věnované API6. Komplikace při vývoji obou řešení jsou shrnuty v kapitole Vyhodnocení9. Závěr10 zhodnocuje úspěšnost celého projektu a zamýšlí se nad budoucím vývojem.

1.1 ARTIO

Současná podoba společnost ARTIO s.r.o. se začala tvořit v roce 2002, kdy došlo k zaměření na vývoj software a později vývoj a nasazení open source webových systémů. Stejně jako zaměření společnosti se měnil i počet pracovníků. V současné době spolupracuje s přibližně deseti zaměstnanci či externisty. Mezi hlavní činnosti patří tvorba webových prezentací, implementace open source řešení, vývoj modulů a pluginů pro Joomla! a Magento.

2 Joomla

Tato kapitola pojednává o jednom z klíčových technických prostředí, ve kterém se ARTIO prosazuje. V úvodu jsou popsány jednotlivé stavební prvky tohoto systému, v druhé kapitole jsou stručně shrnuty nejdůležitější verze a historie Systému. V poslední podkapitole jsou popsány komerční komponenty nabízené společností ARTIO.

Joomla! je úspěšný systém pro správu obsahu (CMS) 2.1, postavení na skriptovacím jazyku PHP. Joomla dostala v roce 2011 ocenění od Packt Publishing award[1] pro nejlepší open source CMS systém. Podobného ocenění se jí dostalo i v letech 2006 a 2007.

Definice 2.1 *CMS - redakční systém: systém pro správu obsahu, pomocí kterého je možné kompletně spravovat veškerý obsah webové stránky bez znalostí programování či jiných odborných technických znalostí.*

V dubnu roku 2012 Joomla dosáhla významného milníku. V tuto dobu přesáhlo celkové stažení tohoto redakčního systému 30 miliónů stažení[2]. Odhady z této doby předpokládají, že Joomla běží na 2.7% všech webových stránek na internetu. Mezi stránky, které využívají CMS Joomla nepatří jen malé osobní stránky, ale i webové prezentace a intranety nadnárodních korporací jako jsou Ikea, McDonald's, Harvard University, General Electric, eBay nebo třeba Citibank. Joomla je rozšířená i ve vládním a vojenském sektoru[3]. Velkou předností Joomly je možnost snadno dodat speciální funkcionalitu pomocí extenzí.

2.1 Extenze

Je nutné uvést, že zde uvedené názvy různých typů extenzí jsou doslovně přeložené z oficiálního anglického popisu[4]. Z pohledu informatika zde dochází k výrazné záměně pojmů. Originální názvy slouží hlavně administrátorům Joomly u kterých se nepředpokládá znalost programování nebo inženýrství.

Komponenta je zpravidla větší a komplexnější rozšíření, který by mělo být nezávislé na ostatních komponentách. Komponentu je možné si rovněž představit jako malou aplikaci běžící v rámci redakčního systému. Komponenta se stará a zobrazení dat na jakékoliv stránce. Systém po instalaci obsahuje základní komponenty, které obstarávají například zobrazování/editaci článků nebo uživatelů. Každá komponenta je rozdělená do dvou částí. Veřejná část je dostupná na hlavní stránce. Druhá, administrační část je dostupná pouze z administrace a slouží k nastavení komponenty.

Modul je menší prvek, který může být vložený na libovolnou stránku. Jeho funkcionalita je jednoduchá a neobsahuje komplexní logiku (někdy se dokonce jedná pouze o HTML kód). Modul může využit pro zobrazení menu, patičky stránky, předpovědi počasí, či jen reklamního banneru. Některé odkazují na konkrétní komponenty, či z nich zobrazují data.

[1] http://www.joomla.org/announcements/general-news/5394.html
[2] http://www.joomla.org/announcements/general-news/5421.html
[3] http://community.joomla.org/showcase/
[4] http://docs.joomla.org/Extension_types_(general_definitions)

Plugin reaguje na události ke kterým je registrován. Plugin není určen k zobrazení informací jako modul, ale je určen k provedení určité akce při konkrétní události. jedná se téměř o jediný prostředek pomocí kterého můžeme reagovat a zpracovávat akce provedené jádrem Joomly nebo jakoukoliv extenzí.

Šablona slouží pro renderování HTML stránky. Nahráním nové šablony můžeme kompletně změnit vzhled webu. Do šablony se při renderování navštívené stránky načítá komponenta a na patřičné místo se vkládají asociované moduly.

Jazykové rozšíření je tou nejjednodušší formou extenze. Jedna se o textový soubor obsahující překlady ve formátu klíč="hodnota". Název souboru je specifický vzhledem k jazyku pro něhož se má soubor použít. Tato extenze může přidávat překlad pro jádro Joomly nebo například pro modul nebo komponentu. Je tak možné jednoduše spravovat a vytvářet jazykové mutace webu.

2.2 Verze

Joomla prošla od svého vzniku velkým vývojem. Od prvních verzí, které byly funkcionálně obsáhlé, ale velice složité na ovládání, až po současné verze, které jsou responzivní a uživatelsky daleko více přívětivé. Následující výčet nejdůležitějších verzí poskytne strohou představu o postupném vývoji tohoto redakčního systému.

Joomla 1.5.0 [5] vydaná v lednu roku 2008 přinesla hlavně zjednodušení administrace systému. Administrace byla podstatně zjednodušena a dostupná v lokalizované jazykové podobě.

Joomla 1.6.0 [6] vydaná v lednu roku 2011 představovala velký krok vpřed. Mezi hlavní funkce, které byly přidány patří podpora pro komplexní řízení oprávnění uživatelů, zavedení více úrovní uživatelů a zjednodušení instalace extenzí. Externí extenze je tak možné instalovat doslova jedním kliknutím myši. Pro vývojáře komponent byl velkým přínosem zavedení architektury MVC, která umožnila přehlednější a rychlejší vývoj komplexních komponent.

Joomla 2.5.0 [7] vydaná o rok později přinášela převážně bezpečnostní záplaty, ale i tak obsahovala na 26 nových funkcí. Třebaže tato verze funkcionálně není tak převratná jako verze 1.6.0, dočkala se obrovského rozšíření.

Joomla 3.0.0 [8] vydaná v září roku 2012 přinesla, stejně jako verze 1.6.0, velké změny napříč celým systémem. Nejvýraznější je integrace s Twitter Bootstrap a předělání základního layoutu dle mobilních a responzivních požadavků na stránky. Méně viditelné, ale zato kritičtější změny nastaly v jádru frameworku. Migrace existujících komponent pro verzi 2.5.0 byla velice jednoduchá, ale komponenty pro verzi Joomly 1.6.0 prakticky nebylo možné efektivně migrovat.

[5] http://www.joomla.org/announcements/release-news/4483.html
[6] http://www.joomla.org/announcements/general-news/5348.html
[7] http://www.joomla.org/announcements/release-news/5403.html
[8] http://www.joomla.org/about-joomla.html

2.3 ARTIO komponenty pro Joomla

Společnost ARTIO[9] vyvíjí několik placených komponent, které je možné jednoduše integrovat do Joomly a obohatit tak systém o požadovanou funkcionalitu. Tyto komponenty jsou dostupné v demo verzích a taktéž v licencovaných verzích, které obsahují rozšířenou funkcionalitu. Možnost updatu na novou verzi a zákaznická podpora jsou dostupné po dobu platnosti zakoupené licence. Licenci lze vždy prodloužit za zvýhodněných podmínek. Aktuálně nabízené komponenty pro Joomla jsou:

- Bookit!
- JoomSEF
- JoomDOC
- FusionCharts
- E-Tickets
- VM Invoice

Mezi nejúspěšnější a taktéž nejkomplexnější komponenty patří převážně Bookit! a JoomSEF.

Bookit! je komponenta pro online rezervace. Komponentu používají hotely, autopůjčovny či zubní ordinace. Časté je rovněž nasazení na menších penzionech, tenisových kurtech a konferenčních místnostech. Jednotlivé rezervovatelné události lze natolik specifikovat a upravit, že prakticky neexistuje typ rezervace kterou by tento systém nebyl schopen obstarat. Mezi hlavní výhody komponenty patří kromě univerzálnosti i přehledný kalendář pro rezervaci, detailní nastavení cenové politiky a napojení na nejpoužívanější platební brány. V současné době se jedná pravděpodobně o nejkomplexnější Joomla komponentou tohoto druhu na trhu.

JoomSEF komponenta se zaměřuje na SEF URL adresy webu a generování relevantních meta tagů. Joomla umožňuje generování "pěkných"URL, ale neumožňuje další detailnější nastavení, či vlastní pravidla. Komponenta JoomSEF poskytuje komplexní nástroj pro správu a generování SEF URL, stejně jako automatické vkládání meta tagů k patřičným URL adresám. Postará se i o přeložení URL adres pro jednotlivé jazykové mutace webu. V současné době se jedná pravděpodobně o nejrozšířenější komponentou tohoto druhu na trhu.

[9] `http://www.artio.net`

3 Současné řešení

V této kapitole je popsána současná situace ve společnosti ARTIO z pohledu workflow pro Joomla komponenty a taktéž je popsán informační systém využívaní pro projektový management Redmine.

Jak už bylo uvedeno v Úvodu1, současně je společnost téměř zahlcena vyřizováním uživatelské podpory. Tato uživatelská podpora je většinou vyřizovaná v rámci podpory k zakoupené licenci produktu a stojí čas, tím pádem i nemalé finanční zdroje. V ojedinělých případech může nastat situace, kdy náklady na komunikaci s uživatelem a řešení daného problému téměř dorovnají nebo dokonce překročí cenu zakoupené komponenty (licence). Komponenta Book it! stojí v této chvíli 49 Euro. Cena zahrnuje plně funkční komponentu a licenci na jeden rok, která opravňuje k updatu na nově vydané verze a zákaznickou podporu. Jinými slovy částka 49 Euro musí bezpečně pokrýt spoluúčast na ročním vývoji komponenty a veškerou komunikaci s klientem včetně řešení problémů s nastavení komponenty a serveru.

Pokud má klient nějaký dotaz ohledně komponenty, či problém nebo hlášení chyby, využije diskuzní fórum, kde v klientské sekci a v kategorii dané komponenty vloží dotaz na který někdy reaguje sekretářka nebo většinou přímo vývojář dané komponenty. Výhodou fóra je přehlednost všech dotazů a rychlost reakce. Hlavní nevýhodou je, že pokud je problém složitější povahy a je nutný přístup do systému klienta, přístupové údaje se musí posílat neveřejnou cestou - emailem a tím se komunikace rozděluje mezi fórum a email, a stává se méně přehlednou.

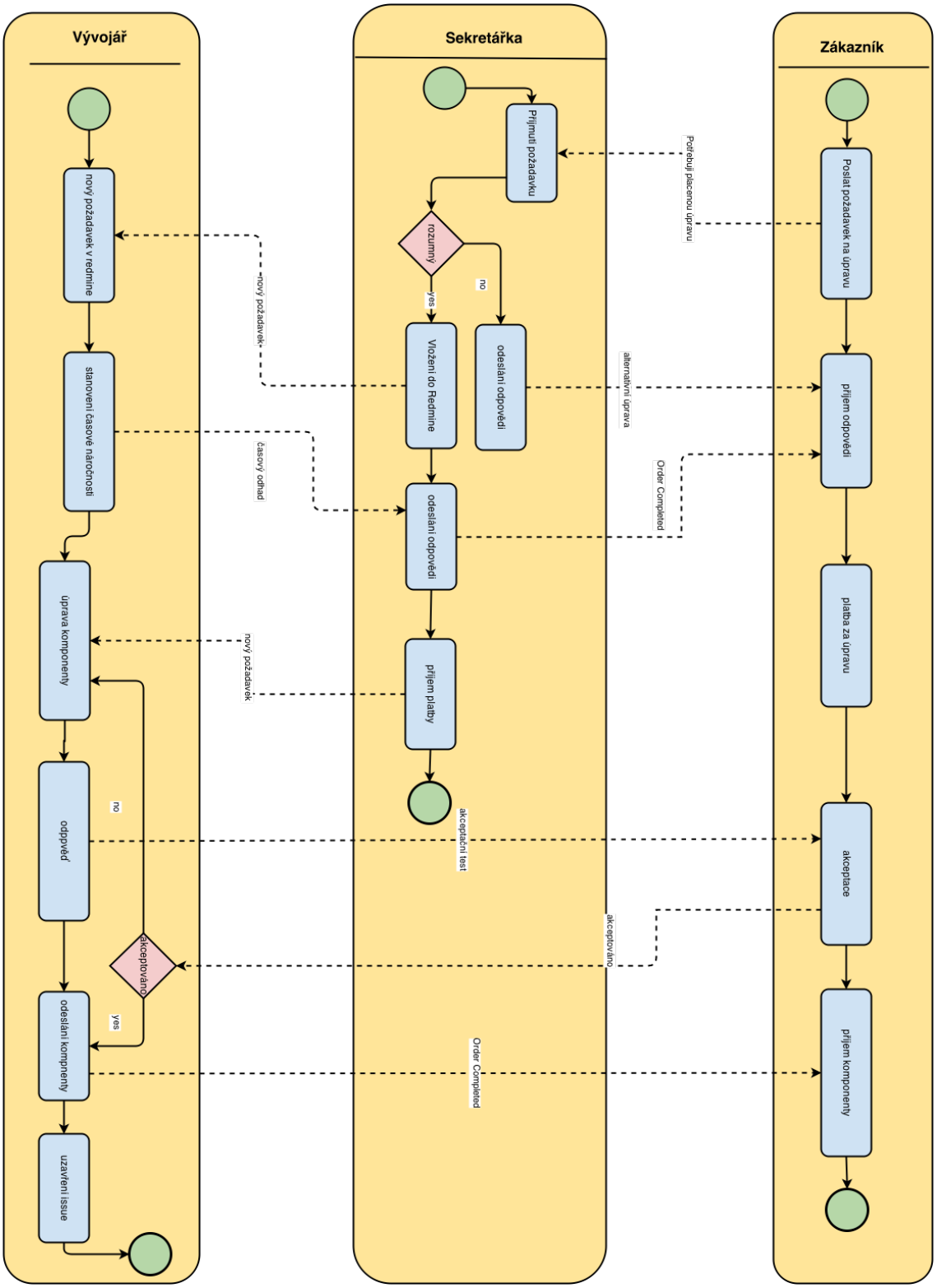
Některým klientům zcela nevyhovuje zakoupená komponenta a požadují dodatečnou, placenou úpravu dle jejich požadavků. Takový požadavek nazýváme placená podpora. Zákazník s konkrétním požadavkem kontaktuje společnost pomocí emailu. Sekretářka vykomunikuje s programátorem očekávanou dobu potřebnou pro implementaci funkcionality a odpoví klientovy s uvedenou časovou náročností úkolu. Pokud klient souhlasí s rozsahem práce, zakoupí potřebný počet "support"hodin přes svůj účet na artio.net. Po provedení platby sekretářka zpracuje emailovou komunikaci s klientem a vloží ho do interního systému Redmine jako nové issue přiřazené ke konkretnímu vývojáři. Programátor se snaží implementovat požadavek jako změnu konkrétní funkce komponenty nebo přidáním zcela nové funkce. Komponenta je instalovaná na interní server, na kterém je vytvořen testovací účet pro klienta, který komponentu odzkouší a případně připomínkuje. Po konečném schválení je komponenta poslaná klientovi, v některých případech zároveň instalovaná na klientův Joomla systém.

Nová verze komponenty obsahuje opravené chyby, nové funkce nebo úpravy a funkcionalitu implementovanou v rámci placené podpory. Tento systém vývoje umožňuje částečně financovat nové funkce komponenty z placené podpory. Uživatelé, kteří zaplatili za speciální úpravu pak budou mít možnost přejít na novou verzi komponenty, která splňuje jejich speciální požadavky. Komponenta se tak stává mnohem komplexnější, univerzálnější ale i obtížnější na údržbu a testování. Tímto přístupem je možné efektivně vyvinout "enterprise"komponentu s největší funkcionalitou na trhu. S rostoucím počten klientů roste zájem o specifické úpravy, které vylepšují komponentu. Zároveň se stává

nastavení komponenty složitější a počet požadavků na podporu v rámci licence roste rychleji než počet aktivních licencí.

Někteří malí klienti jsou velice překvapení, když zjistí ceny za požadovanou úpravu. Drobná jednohodinová úprava stojí stejně jako celoroční licence na produkt. Běžná úprava pak několikanásobně překračuje cenu komponenty, což je pro drobné zákazníky často neakceptovatelné a tak odstoupí od svého požadavku.

3.1 Workflow



Obrázek 1: BPMN diagram

3.2 Redmine

Redmine[10] je populární [11] [12] open source webová aplikace pro projekt management. Ani není divu, že tenhle flexibilní systém postavený na Ruby on Rails frameworku používá třeba i samotné Ruby[13].

Projekt je hlavní položkou v Redmine. Projektů může být několik každý projekt může obsahovat další vnořené projekty. Projekt by měl představovat logicky i technicky nezávislý celek. V ARTIO je například projekt Joomla Extensions obsahující podprojekty pro jednotlivé komponenty. Komponenta Book it! má svůj projekt a může být tedy spravována a vyhodnocována bez závislosti na ostatních projektech.

Issue je hlavním stavebním prvkem projektu. Issue představuje problém, či úkol, který chceme v rámci projektu řešit. Issues mohou být rozděleny podle typu, zde nazvané jako Tracker. Toto rozdělení specifikuje workflow práce s issue. Každý Tracker má přesně definovanou posloupnost stavů ve kterém se může issue nacházet. Povinnými položkami v issue jsou Tracker, Subject, Status, Priority. ARTIO používá Asignee k přiřazení konkrétního uživatele k danému issue. Due date je datum do kterého má být issue uzavřeno a Estimated time je součet všech strávených časů pro dané issue.

Mezi dodatečné informace v issue patří Category, Target version, Parent task, Start date, % Done, Watchers.

Plán slouží k plánování vydání nových verzí. Zobrazuje verze a k nim přiřazené issues. Ke každé verzi je zobrazeno kolik % issue je ve stavu Closed, tedy dokončeno. Tento výpis je užitečný pro zjištění zda je projekt ve stavu vhodném pro vydání nové verze.

Gantt diagram zobrazuje průběh jednotlivých issue v rámci projektu. Jako ukazatel dokončení pro issue je brána hodnota % Done, která však v malých issues nebývá vývojáři vyplňována a tak ten diagram nemá téměř žádnou vypovídací schopnost.

Kalendář přehledně zobrazuje issue se Start date a Due date v daný den. Přehledně je tak možné vidět jak přibývají issue, ale konkrétní přínos není znatelný.

Repository umožňuje napojení na SVN, CVS a další často používané verzovací systémy.

API Redmine umožňuje přístup k datům v systému. API je detailněji popsáno v kapitole 6 a jedná se o stěžejní prvek celé této práce.

[10] `http://www.redmine.org`
[11] `http://www.redmine.org/projects/redmine/wiki/WeAreUsingRedmine`
[12] `http://blog.zvikico.com/2010/06/tidbits-from-the-eclipse-community-survey-2010.html`
[13] `https://bugs.ruby-lang.org/`

4 Databáze

V této kapitole dochází k porovnání po desetiletí ověřených relačních databází a moderních NoSQL. Cílem kapitoly je přehledně vysvětlit hlavní rozdíly mezi těmito databázemi a popsat odlišný pohled na datovou vrstvu.

Denně používáme na počítači desítky, možná i stovky aplikací. Pokud chceme uložit aktuální sofistikovanější data v aplikaci, velice často se k tomuto účelu využívají databáze. Tento přístup je nejpatrnější na webových aplikacích. Například při odeslání komentáře pod zajímavým článkem předpokládáme, že se komentář někde uloží a zůstane u příspěvku napořád. Pod pojmem "někde uloží"se obvykle skrývá relační databáze. Stejné řešení využívají i desktopové aplikace[14] pro ukládání nejrůznějších dat. Databáze se proto staly neodmyslitelnou součástí skoro každé počítačové aplikace a mnohdy taktéž jejich základním kamenem.

4.1 RDBMS

Relační databáze je netřeba nějak zdlouhavě a objevně vysvětlovat. Jsou mezi námi už přes 20 let a snad každý programátor musí umět alespoň základní práci s těmito databázemi. Relační databáze je označení začínané používat v roce 1970[15]. Od té doby se relační databáze vyvíjí a zdokonalují. RDBMS prakticky dominují databázovému trhu a open source databáze jako MySQL tvoří základ mnoha internetových stránek. Tyto databáze jsou vhodné zejména pro ukládání dat se silnou několikanásobnou vazbou.

ACID

RDBMS databáze dodržují pravidla představující zkratku ACID.

- Atomicita (Atomicity) zaručuje, že transakce (dotaz nebo skupina dotazů uzavřených v transakci) bude provedená kompletně nebo vůbec.
- Konzistence (Consistency) znamená, že databáze bude vždy v "korektním stavu".
- Izolovanost (Isolation) každá transakce dostane výhradní přístup k datům a nikdo jiný nemůže provádět operaci se stejnými zdroji ve stejný čas. Tohoto se dosahuje pomocí sériového vykonání.
- Trvanlivost (Durability) zaručuje trvanlivost dat. Pokud je transakce dokončená, data musí být korektně zapsáni i kdyby došlo k chybě a pádu databáze.

Ačkoliv mnoho moderních RDBMS databází fakticky dokonale nesplňuje všechny ACID požadavky, jsou i přesto označované za ACID. Na problém ohledně izolace dle standardu ANSI SQL-92 poukazuje dokument [4].

Relační mánie

[14] `https://sqlite.org/famous.html`

[15] `http://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf`

Jak už bylo uvedeno v úvodu, databáze je důležitá část každé aplikace či systému. Vhodně navrhnout strukturu relační databáze může být velice obtížný úkol. Všechny tabulky a sloupce se musí navrhnout co nejlépe, aby splňovaly požadavky výsledné aplikace (a někdy i určitou normální formu), jelikož každá dodatečná změna databáze je velice problematická. Dalším obtížným krokem je mapování objektů na databázi. Z těchto důvodů se často návrh databáze provádí úplně jako první věc na začátku vývoje a posléze na tomto modelu roste daná aplikace. Doménový model, reprezentovaný třídami, pak může věrně kopírovat relační model a opomíjet některé širší souvislosti. Díky tomuto přístupu systém může často působit nepřívětivě pro uživatele, jelikož tak trošku připomíná pohled na tabulky relační databáze. Návrh struktury relační databáze a mapovaní objektů (ORM) doménového modelu na tuto databázi je natolik složité a kritické, že méně zkušení vývojáři tyto činnosti neadekvátně prioritizují. Více se o těchto problémech pojednává v [1, 2, 3].

4.2 SQL

SQL je strukturovaný dotazovací jazyk navržený speciálně pro relační databáze a zároveň se jedná o nejrozšířenější dotazovaní jazyk pro tyto databáze. Jelikož jsou pojmy SQL a RDBMS velice úzce spojeny, často dochází k zaměňování pojmů a používání těchto slov jako synonym. Z dnešního pohledu je důležité si uvědomit, že SQL jazyk vznikl pro potřeby počítačů před 40 lety a je často přirovnáván k jazyku COBOL. Vědci potřebovali předat počítači instrukce, jaká data má nalézt. Naprostá většina programovacích jazyků (snad jen kromě C) z té doby už fakticky zanikly[16], stejně jako hardware. Avšak například jazyk COBOL je dnes stále používán při většině[17] světových finančních operací.

Na světě jsou miliardy řádků kódu napsaném v COBOL[18], které je potřeba udržovat. Existence programů napsaných v COBOL ale bohužel nic nemění na skutečnosti, že tento jazyk je již dávno překonán a nové aplikace se vyvíjejí s použitím mnohem modernějších programovacích jazyků.

Dotazovací jazyk SQL, stejně jako COBOL, přečkal desetiletí. Tento jazyk, vytvořený pro potřeby historických počítačů, bohužel není příliš přívětivý pro programátora[19]. Avšak na rozdíl od COBOL, nové aplikace stále využívají SQL (z důvodu používání RDBMS databází). Ačkoliv existuje mnoho abstrakci nad databázovou vrstvou, které odizolují programátora od SQL, komplexní (složité a špatně čitelné) dotazy se většinou musejí psát v SQL. Agregovat vazbu M:N pomocí SQL vyžaduje několik řádků kódu s několika JOIN příkazy, eventuálně dalšími vnořenými dotazy. Při objektově orientovaném programování a dodržování zásad psaní přehledného kódu se snažíme vytvářet co nejkratší a nejčitelnější třídní metody. Některé pokročilejší SQL dotazy jsou však natolik dlouhé a nepřehledné, že snižují čitelnost kódu a znesnadňují jeho údržbu.

Ukázkové SQL dotazy z výpisů 1, 2 a 3 jsou zcela běžné ve webových aplikacích. Jde o mírně pokročilé dotazy, které by měl ovládat každý kdo s RDBMS pracuje. I tyto jedno-

[16] http://readwrite.com/2012/06/05/5-ways-to-tell-which-programming-lanugages-are-most-popular
[17] http://cis.hfcc.edu/faq/cobol
[18] http://www.computerworld.com/s/article/266228/Cobol_Coders_Going_Going_Gone_
[19] http://www.geocities.com/tablizer/sqlcrit.htm

duché dotazy mohou snížit čitelnost kódu. Z vlastní zkušenosti mohu říct, že není příliš velký problém najít v aplikacích dotazy mnohem delší a sofistikovanější. Takové dotazy je nutné zdlouhavě "rozšifrovávat"abychom pochopily jak fungují a co je jejím cílem. Pokud je takový komplexní SQL dotaz napsán nesprávně a vrací jiné než požadované výsledky, oprava je velmi zdlouhavá a náročná. Bohužel v rámci SQL téměř neexistuje efektivní způsob, jak se vyhnout těmto složitým dotazům. Tento popis nemá sloužit jako kritika SQL jazyku, mým cílem je pouze poukázat na tyto vlastnosti.

```
"SELECT s.close AS original,
(
    select ss.close
    from `# __etickets_event_static_schedule ` ss
    where ((ss.week_day = ".$day.") OR (ss.week_day = 0)) and ss.valid_from <= ".$fdate." and ss.
        valid_to >= ".$fdate."
    limit 1
) AS override
FROM `#__etickets_event_static_schedule` AS s
WHERE s.week_day = ".$day." and s.valid_from IS NULL and s.event_id=".$eventId.""
```

Výpis 1: Ukázkový SQL dotaz 1

```
"SELECT oi.order_status, t. event_start_validity , t . event_end_validity , t . event_admission, pt.
      etickets_start_validity as begin_action, pt . etickets_end_validity as end_action, pt .
      etickets_ticket_template , oi . order_item_id , pr . product_name as order_item_name, oi.
      product_quantity, oi. product_final_price , oi . order_item_currency, oi . product_attribute , pr .
      product_url , oi . cdate, pt . owner
FROM #__etickets_event t
JOIN #__{vm}_order_item oi ON (t.order_item_id=oi. order_item_id)
JOIN #__{vm}_product prp ON (oi.product_id=prp.product_id)
LEFT JOIN #__{vm}_product pr ON (prp.product_parent_id=pr.product_id)
JOIN #__{vm}_product_type_".$this->event_product_type_id." pt ON (prp.product_parent_id=pt.
    product_id)
WHERE (t.event_ean='$this->ean')"
```

Výpis 2: Ukázkový SQL dotaz 2

```
"SELECT t.ticket_id, t . subject , t . status , t . local_status , t . estimated_hours, t . fund_end, IFNULL(p
    .ticket_id , t . ticket_id ) AS users, sum(p.max_hours) AS bid_hours, pp.max_hours as my_bid
FROM #__asupport_ticket AS t
LEFT JOIN #__asupport_participant AS p ON t.ticket_id = p. ticket_id
LEFT JOIN #__asupport_participant AS pp ON t.ticket_id = pp. ticket_id AND pp.user_id = ".$this->
    user->id."
WHERE t.type= '".AsupportAccount::TYPESHAREDDEV."'
GROUP BY users
ORDER BY t.updated DESC"
```

Výpis 3: Ukázkový SQL dotaz 2

4.3 NoSQL

NoSQL databáze umožňují dotazovat se na data nejen pomocí SQL (Not-Only SQL). Hlavní rozdíl je však téměř úplná eliminace relací mezi daty pro některé databáze typu NoSQL. Tyto databáze se zjednodušeně dělí dle [6] na

- Dokumentové
  Každý záznam představuje komplexní strukturu nazvanou dokument. Dokument může obsahovat záznamy typu klíč-hodnota, ale i pole (kolekci) těchto záznamů, popřípadě i jiný vložený dokument. Struktura těchto záznamů není nijak definovaná a proto se jedná o takzvané schema-less (schema-free) organizaci dat. Dokument bývá uložen v JSON formátu.
- Grafové
  Grafové databáze jsou určeny pro uchování mnoha vazeb mezi položkami. Typicky se jedná o sítě, grafy, nebo třeba propojení na sociálních sítích.
- Klíč-hodnota
  Jedná se o nejjednodušší NoSQL databáze, které ukládají záznamy pouze typu klíč-hodnota. Jsou natolik jednoduché, že jejich omezení umožňují jejich použití pouze ve specializovaných případech, kde dosahují obdivuhodného výkonu.
- Se širokými sloupci
  Tyto databáze umožňují dynamicky přidat velké množství (milióny) sloupců pro každý záznam. Dle [5] jsou společné sloupce uloženy pospolu namísto klasického uložení po řádcích. Lze o nich prohlásit, že jsou schema-less, avšak velice odlišné od dokumentových databází. Jsou vhodné zejména pro rychle hledání a třídění podle sloupců.

4.3.1 CAP theorem

Existují tři základní požadavky na distribuovaný systém. Tyto požadavky jsou označovány zkratkou CAP.

- Konzistence (Consistency)
- Dostupnost (Availability)
- Tolerance rozdělení (Partition Tolerance)

Mezi těmito požadavky existuje závislost nazvaná jako CAP teorém (více o závislosti v [7]). U distribuovaných systémů je vždy nutné upřednostnit dva požadavky na úkor třetího. Nikdy nemůže být dosaženo stavu, kdy je systém naprosto konzistentní (atomický), vysoce dostupný (bez velké latence) a odolný vůči výpadkům dílčích distribuovaných částí[20]. Vždy je třeba přijmout riziko a navrhnout systém tak, aby případné problémy

[20] `http://www.julianbrowne.com/article/viewer/brewers-cap-theorem`

CAP řešil co nejspolehlivěji.

NoSQL databáze oproti RDBMS databázím, a taktéž díky CAP teorému, mají vlastnosti shrnuté jako "BASE":

- Běžná dostupnost (Basically Available)
- Měnitelný stav (Soft state)
- Většinová konzistence (Eventual consistency)

Tyto vlastnosti v praxi znamenají, že občas můžeme obdržet z databáze neaktuální data. Jinými slovy, že uživatel v Americe může obdrží hodnotu A, zatímco uživatel v Evropě může na stejný dotaz obdržet hodnotu B. Více o tématu lze najít v článku The Challenges of Latency [8], případně v [9, 10]. Toto platí pouze pro distribuované systémy, kde škálovatelnost NoSQL je daleko lepší než škálovatelnost RDBMS databází. Pokud budeme provozovat pouze jednu databázi na jednom serveru, této občasné inkonzistence dat se samozřejmě nemusíme obávat.

4.3.2 Důvody pro NoSQL

Škálovatelnost

Škálovatelnost je dvojího typu. První je vertikální, druhá se nazývá horizontální. Vertikální škálovatelnost představuje navýšení výkonu počítače. V praxi se jedná výměnu procesoru za rychlejší model, nebo přidání operační paměti. Vertikální škálovatelnost je limitována technologií, protože například není technicky možně, aby měl jeden počítač 1TB operační paměti. Náklady na zvyšování výkonu přitom nerostou lineárně, ale spíše exponenciálně. Horizontální škálovatelnost přidává další počítače o stejném výkonu, přičemž se zátěž rozdělí mezi tyto počítače (shardy). V praxi můžeme mít například 16 počítačů s operační pamětí o kapacitě 64GB (8x8GB). Velkou výhodou NoSQL databází je, že jsou jednoduše vertikálně škálovatelné. Data mohou být rozdělena na několik shardů, jelikož neexistují téměř žádné vzájemné vazby mezi záznamy. Praktické provedení vypadá například tak, že se část zákazníků žijící v USA přenese na jeden shard, zatímco zákazníci z EU budou na odlišném shardu.

Flexibilita

Jako dobrý příklad poslouží MongoDB. Tato dokumentová databáze ukládá data ve formátu JSON. Jednotlivé dokumenty v databázi nejsou nijak omezeny svou strukturou a díky ukládání v JSON formátu je možné přímo ukládat serializované objekty nebo složité struktury. Změna existujícího dokumentu taktéž není omezena a proto je možné dokument rozšiřovat či měnit dle doménového modelu, bez ohledu na databázi. Tato vlastnost je velice příjemná při vývoji, kdy se nemusí monotónně měnit struktura databáze při každé nové funkcionalitě. Daleko kritičtější využití nastává na již nasazené aplikaci, kdy dochází ke snaze minimalizovat zásah do databáze. Při změně programu tak nedochází k úpravě databáze, pouze ke změně v ukládaných datech. Krátký úvod do NoSQl lze najít na stránkách databáze CouchDB[11], užitečná může být i infografika[12].

4.4 SQL nebo NoSQL

Každá databáze má svůj záměr použití. Jelikož je RDBMS standardem, otázka spíš zní za jakých okolností zvolit jiné řešení, tedy NoSQL. Situací kdy zvolit NoSQL je až překvapivě mnoho ale popíšeme zde jen několik zásadních.

Struktura dat je z vývojářského pohledu důležitý aspekt při rozhodování. Data která jsou vícerozměrná nebo se složitou či neznámou strukturou je vhodnější uložit do NoSQL dokumentové databáze, jelikož mapování těchto dat na RDBMS je příliš komplikované v porovnání s ukládáním serializovaných objektů v NoSQL.

Způsob vývoje aplikace , kdy je využíváno agilních technik je rovněž téměř předurčen pro databáze bez definované struktury dat. Při iterativním vývoji se každý týden (perioda jednoho sprintu) přidávají nové funkce a aplikace musí být vždy funkční. Dopředu je nemožné určit výslednou strukturu dat a změny RDBMS při vývoji nebo běhu systému jsou obtížné a časově náročné.

Výkon bývá zmiňován vždy na prvním místě, ale je třeba si uvědomit, že pro velké množství aplikací by bylo lepší použít NoSQL databázi ze dvou předešlých důvodů, bez ohledu na to zda aplikace trpí výkonovými nedostatky. Nicméně pro aplikace, které vyžadují masivní škálovatelnost, je vyžití NoSQL databází velice vhodnou volbou neboť je možné je snad horizontálně škálovat.

Pokud máme stávající řešení postavené na běžné RDBMS jako je MS SQL nebo MySQL, pak je pravděpodobně možné použít NoSQL dokumentovou databázi[21].

Obecně řečeno: Běžně používanou RDBMS databázi jako je MySQL lze nahradit NoSQL dokumentovou databází, například MongoDB. Smysluplnost tohoto rozhodnutí (viz. [13]) je však nutné důsledně zvážit. Pokud náš doménový model neobsahuje velké množství izolovaných celků sdílených mezi několika prvky, je vhodné použít NoSQL dokumentovou databázi. Návrh datové struktury pro NoSQL databáze je sám o sobě docela jednoduchý a přímočarý. Databáze nám přesně nediktuje jak máme data ukládat a datové schéma si můžeme vytvořit svobodně sami.

Ačkoliv jsou NoSQL databáze na vzestupu a například MongoDB se začíná masivně prosazovat[22], v České Republice aktuálně neexistuje žádný poskytovatel sdíleného webhostingu, který by nabízel MongoDB databázi, což znatelně brání v masivnějšímu rozvoji.

[21] `http://www.mongodb.com/learn/nosql`
[22] `http://db-engines.com/en/ranking`

5 Vize nového řešení

Jak bylo uvedeno v kapitole Současné řešení3, stávající situace doslova ohrožuje chod společnosti. Vize vychází z těchto problémů a pokouší se popsat ideální řešení.

Nekritičtější problém je velký počet dotazů na uživatelskou podporu. Minimalizace těchto mnohdy zbytečných dotazů povede k ušetření nemalých nákladů. Tým vývojářů se pak bude moct lépe soustředit na zvyšování kvality poskytovaného softwaru namísto nastavování komponent u klientů.

Dlouhodobým cílem je zvýšit množství funkcionality implementované v rámci placených požadavků od zákazníků. Navýšení poměru externě a interně financovaných úpravu je v přímé souvislosti s tímto požadavkem. Tento krok sám o sobě přinese okamžité zvýšení zisku. Z dlouhodobého pohledu se jedná o efektivní řešení pouze v případě úspěšného splnění předchozího požadavku.

Zefektivnění projektového řízení v návaznosti na Redmine je klíčové hlavně v případě úspěšného zvládnutí předchozího bodu. Projekty financované externě jsou vždy náročnější na řízení než běžné interní projekty. V případě nízkého počtu externích projektů fungují interní projekty jako jakýsi buffer. V případě zpoždění je často vhodnější upřednostnit dokončení externí zakázky na úkor interního projektu. Pokud je poměr externě a interně financovaných úprav vysoký, buffer v podobě interních projektů se zmenšuje a opožděné projekty není možné dokončit včas. Opoždění interního projektu o dva týdny je většinou pouze nepříjemné, ale nijak zásadní. Zpoždění zakázkové úpravy komponenty o dva týdny je mnohem závažnější a jednoznačně uškodí dobrému jménu společnosti. V tomto případě je nutné klást velký důraz na řízení projektu a dopad strategických rozhodnutí. Redmine neposkytuje dostatečné nástroje pro podporu řízení projektů. Vhodný analytický nástroj by pomohl manažerům v rozhodování a prioritizování jednotlivých issue.

Z vize je patrné, že by bylo vhodné pokusit se implementovat dva separované systémy. Jeden systém na správu požadavků od klientů a druhý analytický pro podporu rozhodování manažerů. Jelikož se výsledné řešení rozpadlo na dva dílčí a nezávislé celky, od této chvíle budeme tuto skutečnost respektovat a vývoj obou řešení bude probíhat naprosto separovaně.

6 Redmine API

V této kapitole jsou popsány základní API přístupy REST a SOAP. Dále je popis s ukázkami kódu pro komunikaci s Redmine API[23], jelikož celá implementace staví na předpokladu, že vyvíjené řešení lze snadno propojit s Redmine.

6.1 REST

Redmine poskytuje funkcionálně poměrně obsáhlé REST API. S API je možné komunikovat skrz formát JSON nebo XML.

REST

REST je architektura rozhraní pro přístup ke zdrojům. REST využívá HTTP protokol a ačkoliv si to někdy zcela neuvědomujeme, web, tak jak ho známe, je postaven na REST. Všechny dnešní webové stránky využívají RESTový přístup k datum. Jedná se o přímý přístup ke zdrojům. Každý zdroj je jednoznačně identifikován URL adresou. akci kterou chceme se zdrojem provést je nutné specifikovat pomocí HTTP metody. Podporované metody jsou

- GET - získání záznamů
- POST - vytvoření nového záznamu
- PUT - změna záznamu
- DELETE - smazání

SOAP nebo REST

Rozdíl mezi SOAP a REST je velice zásadní. SOAP je specifikace protokolu pro výměnu zpráv pomocí XML formátu, kdežto REST je datově orientována architektura (CRUD). Oba dva přístupy využívají HTTP protokol, ale každý velice odlišně. Zatímco REST je na tomto protokolu doslova postaven, SOAP jej využívá pouze částečně.

Následující ukázka[24] představuje stejný dotaz provedený pomocí SOAP a REST. SOAP posílá XML dokument na server, zatímco REST pouze ”zavolá”URL adresu.

[23] `http://www.redmine.org/projects/redmine/wiki/Rest_api`
[24] `http://rest.elkstein.org/`

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:body pb="http://www.acme.com/phonebook">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

Výpis 4: Ukázka SOAP požadavku

```
http://www.acme.com/phonebook/UserDetails/12345
```

Výpis 5: Ukázka REST požadavku

V posledních letech je SOAP na ústupu pro svou těžkopádnost a ve velké míře se prosazuje REST a formát JSON. Určitou vlastností REST (a HTTP protokolu) je, že je bezstavový, což nám v běžných aplikacích (kde převažující operace CRUD) nečiní žádné potíže, ale můžou nastat situace, kdy je stavovost kritická. Naproti tomu SOAP umožňuje ACID v případě transakce, čehož REST není schopen.

Ve většině případů je vhodné použít REST, zvláště pak pro aplikace zaměřené čistě na CRUD. Použití SOAP je doporučeno pouze v konkrétních speciálních případech, kdy preferujeme procedurální přístup. Pokud například chceme provádět finanční transakce přes API banky, ACID vlastnosti, které SOAP splňuje, jsou zde klíčové. SOAP je považován za spolehlivější, díky standardizovaným chybovým stavům (viz. [14]).

6.2 Redmine API implementace

Pro Redmine API existuje několik knihoven, které zjednodušují práci s API. Mezi různorodé podporované jazyky patří Ruby, PHP, Python, Java, .NET a Delphi.

Interakce skrz PHP

Pro připojení k API máme dvě nabízené možnosti. Jedna možnost je využít knihovnu PHP ActiveResource[25], která slouží jako prostředník pro jakékoliv REST API postavené na Ruby on Rails. Knihovna je velice jednoduchá na použití a díky návrhovému vzoru Active Recors je možné ji velice snadno implementovat do již běžícího systému bez nutnosti psaní velkého množství nového kódu.

```
<?php
require_once ('ActiveResource.php');

class Song extends ActiveResource {
    var $site = 'http://localhost:3000/';
```

[25] https://github.com/lux/phpactiveresource

```
    var $element_name = 'songs';
}

// create a new item
$song = new Song (array ('artist' => 'Joe Cocker', ' title ' => 'A Little Help From My
    Friends'));
$song->save ();

// fetch and update, line by line
$song->find (44);
$song->title = 'The River';
$song->save ();

// get all songs
$songs = $song->find ('all');

// delete a song
$song->find (44);
$song->destroy ();

?>
```

Výpis 6: Ukázka ActiveResource

Druhou možností je využít kompletní knihovnu PHP Redmine API[26]. Tato knihovna poskytuje vlastní třídu pro každý zdroj Redmine API. Z tohoto důvodu se jedná a komplexní knihovnu, která však může být velice snadno upravena k vlastním potřebám.

```
<?php

$client = new Redmine\Client('http://redmine.example.com', '1234567890abcdfgh');

//create issue
$client->api('issue')->create(array(
    'project_id' => 'test',
    'subject' => 'test api (xml) 3',
    'description' => 'test api',
    'assigned_to_id' => $userId,
));

//update issue
$client->api('issue')->update($issueId, array(
    'subject' => ''test api (xml) 4',
));

//get list of issues
$client->api('issue')->all(array(
    'limit' => 100
));
```

[26] https://github.com/kbsali/php-redmine-api

```
// delete issue
$client->api('issue')->remove($issueId);

?>
```

Výpis 7: Ukázka Redmine API

Na základě vize se předpokládá, že bude využita téměř veškerá funkčnost poskytována API Redmine a proto se jeví použití této knihovny jako vhodnější.

7 Asupport komponenta

Tato kapitola se pokusí, na základě vize, navrhnout a realizovat nové řešeni. Analýza se zaobírá strukturou a architekturou aplikace, přičemž musí brát v potaz stávající systém pro řízení projektů Redmine. Krátce jsou zmíněny již existující podobné systémy, ale hlavní část je věnována návrhu řešení a konkrétní realizaci.

Jako první je potřeba získat obecnou představu o výsledném řešení. Základní představu o řešeném problému jsme získali z vize, ale je potřeba blíže specifikovat požadavky. FURPS je formalizovaný přepis vize, který obsahuje požadavky na funkce a vlastnosti sytému. Funkce systému jsou rozděleny do scénářů nazvaných případy užití. Každý případ užití je strukturovaně popsány scénář používaní systému tak, aby byl čitelný pro neodbornou veřejnost. V dalších krocích nastupuje modelovací jazyk UML, graficky zobrazující části systému od abstraktních pohledů na komponenty až po detailní komunikací tříd mezi sebou.

7.1 Požadavky

Požadavky je možné specifikovat mnoha různými způsoby. Velmi rozšířené je rozděleni na funkční a kvalitativní (ne-funkční) požadavky. Toto rozdělení slouží jako dobré vodítko při specifikaci požadavků, ale je příliš abstraktní pro většinu systémů. V tomto případě využijeme klasifikační systém FURPS+[27]. První požadavek, jak už z názvu vyplývá, spadá do kategorie funkčních požadavků. Další požadavky jsou převážně požadavky spadajícími do kategorie kvalitativních požadavků.

7.1.1 Požadavky FURPS+

Funkčnost

1. systém umožní vytvořit požadavky typu Ticket, Support, Support(Shared)
2. systém umožní vložit nové požadavky do Redmine
3. systém umožní zobrazit požadavky z Redmine
4. systém umožní vložit komentář k požadavku
5. systém umožní sdílet speciální požadavky typu Support(Shared) mezi uživateli
6. systém umožní zakoupit hodiny vývoje pro požadavek typu Support
7. systém umožní zakoupit tickety pro požadavek typu Ticket
8. systém umožní automatizovat proces platby za požadavky

Použitelnost

1. Komponenta bude graficky odpovídat ostatním komponentám které uživatelé již v rámci systému používají.

[27] `http://www.ibm.com/developerworks/rational/library/4706.html`

2. Proces zadání požadavku bude proveden pomocí jednoho formuláře.
3. Uživatel bude mít vždy přehled o aktuálním stavu jeho účtu a požadavcích.

Spolehlivost

1. Systém bude dostupný 24x7.
2. Každá finanční operace bude zaznamenána v logu.

Výkon

1. Synchronizace dat nebo jiné obdobné procesy budou prováděny v rámci komunikace uživatele se systémem, ale pomocí CRON služby.
2. Komponenta nebude znatelně zpomalovat stávající systém Joomla.

Podporovatelnost

1. Zdrojový kód bude být řádně okomentován.
2. Každá kritická operace bude zaznamenána v logu.
3. Každá kritická chyba bude zaznamenána v logu.

Návrhové požadavky
Je nutné použít stávající RDBMS databázi.

Implementační požadavky
Je nutné použít stávající databázi MySQL systému Joomly. Komponenta musí být napsána v programovacím jazyku PHP a splňovat požadavky pro komponenty systému Joomla 1.5. Měla by v co největší míře využívat návrhového vzoru MVC a dodržet strukturu podobnou ostatním komponentám ARTIO.

Požadavky na rozhraní
Komponenta především musí komunikovat s Redmine API.

Fyzické požadavky
Systém nemá žádné specifické požadavky na hardware.

7.1.2 Základní požadavky na případy užití

Asupport komponenta má na starost zprostředkovat komunikaci mezi pracovníky, kteří používají téměř výhradně systém Redmine a zákazníky, kteří doposud téměř výhradně využívali email. Komponenta má za úkol zpracovávat základní typy dotazů-úkolů. Nejčastější jsou takzvané Ticket dotazy jež jsou k dispozici v rámci licence ke komponentě a mohou být dodatečně dokoupeny. Tyto Tickety většinou představují dotazy uživatelů nebo radu s konkrétním nastavením komponenty. Druhý typ je takzvaný Support, při kterém žádá zákazník o dodatečnou placenou úpravu komponenty. Rozdíl mezi těmito dvěma typy dotazů je pouze malý a proto jsou implementovaný v rámci jedné třídy issue.

Dotaz Support je dále dělen na soukromý a veřejný. Soukromý support dotaz slouží jako "poptávka"po velice specifické úpravě komponenty. Uživatel jednoduše zadá žádost o konkrétní úpravu komponenty. Dotaz je viditelný pouze pro uživatele který ho vytvořil a pro pracovníky z ARTIO. Pokud je dotaz veřejný, opět dochází k malým odlišnostem jako v případě rozdělení na Issue a Support. Veřejný dotaz se stane viditelným i pro ostatní zákazníky a ti se mohou rozhodnout finančně podpořit tuto úpravu libovolně velkou částkou. V praxi to pak znamená, že uživatel který požadavek vytvořil nemusí hradit jeho celý vývoj, ale může se složit s ostatními. Nevýhodou tohoto řešení je výraznější prodloužení implementace, jelikož proces vyžaduje mnohem více času a koordinaci mnoha uživatelů. Vývojář někdy musí přeformulovat požadavek tak aby více odpovídal obecnému zájmu a tak se implementace může mírně lišit od originálního požadavku uživatele. Tento rozdíl je ale vždy pouze z důvodu lepší formulace problému a navrhovaného řešení tak, aby byla úprava zajímavá i pro ostatní uživatele. Tyhle drobně změny se samozřejmě dějí i v případě soukromého požadavku, ale z psychologického hlediska nepůsobí na zadavatele nikterak výrazně.

Základní požadavky na případy užití

- Jako zákazník chci rychle a jednoduše zadat požadavek na úpravu komponenty nebo požádat o radu s nastavením.
- Jako zákazník chci přehledně vést komunikaci o přesnější specifikaci požadavku.
- Jako zákazník chci jednoduše a rychle provést platbu, ideálně stejně jako za zakoupenou komponentu.
- Jako zákazník chci mít možnost finančně podpořit nějakou veřejnou úpravu jiného zákazníka a následně obdržet novou verzi.
- Jako manažer chci získat od uživatele největší množství důležitých informací o požadavku.
- Jako manažer chci přenést všechny požadavky uživatele rovnou do interního systému Redmine.
- Jako manažer chci, aby zákazník zaplatil za požadavek ještě před začátkem práce na požadavku.

7.1.3 Případy užití

Vložení nového Ticketu

Tok činností

1. Zákazník zvolí vytvoření Ticketu na hlavní stránce
2. Uživatel vybere používanou komponentu a její verzi, vyplní zbývající relevantní informace a odešle
3. Systém vloží požadavek do Redmine

4. Systém provede zobrazení požadavku
5. Systém informuje uživatele o úspěšném provedení operace

Rozšíření Nedostatek zakoupených kreditů

1.a Systém zobrazí hlášku o nedostatku zakoupených kreditů
3.a Systém zobrazí hlášku o nedostatku zakoupených kreditů

Rozšíření Nelze vložit issue

3.a Systém zobrazí chybovou hlášku

Zobrazení požadavku

Tok činností

1. Systém načte požadavek z Redmine
2. Systém aktualizuje lokální informace o požadavku
3. Systém zobrazí informace o požadavku

Rozšíření Požadavek neexistuje

1.a Systém přesměruje uživatele na hlavní stránku
1.a Systém zobrazí hlášku o chybě

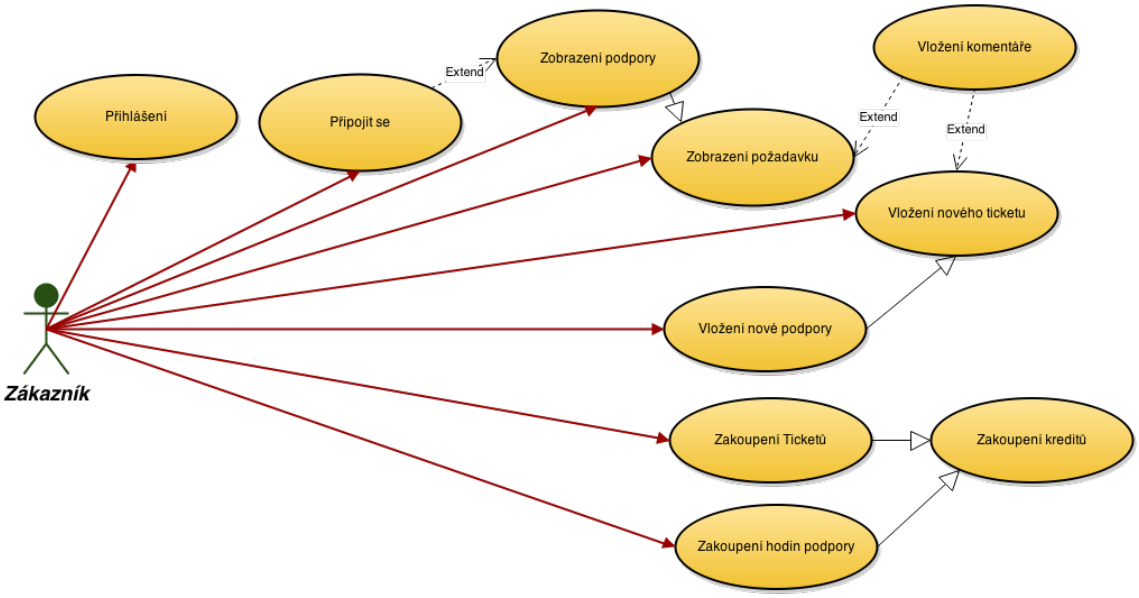
Zakoupení kreditů

Tok činností

1. Uživatel zakoupí požadovaný typ kreditu
2. Systém zpracuje objednávku
3. Uživatel zaplatí objednávku
4. Systém zpracuje zaplacenou objednávku a uživateli navýší počet kreditů

7.1.4 Diagram případů užití

Tento diagram případů užití zobrazuje všechny možnosti, které se naskytují zákazníkovi. Případ užití Zakoupení kreditů z velké části využívá stávající systém (internetový obchod) a proto není dále nikterak rozváděn, avšak proces převodu zakoupených hodin/ticketů z internetového obchodu do toho systému je už v plné režii komponenty. Vložení nového ticketu je téměř totožný s Vložení nové podpory a rozdíly jsou převážně ve formuláři zobrazeném uživateli. Totéž platí o Zobrazení podpory, kdy je oproti Zobrazení požadavku vynechána komunikace mezi zákazníkem a pracovníkem. Při zobrazení podpory je navíc možné Připojit se a částečně tak financovat danou úpravu.



Obrázek 2: Diagram případů užití

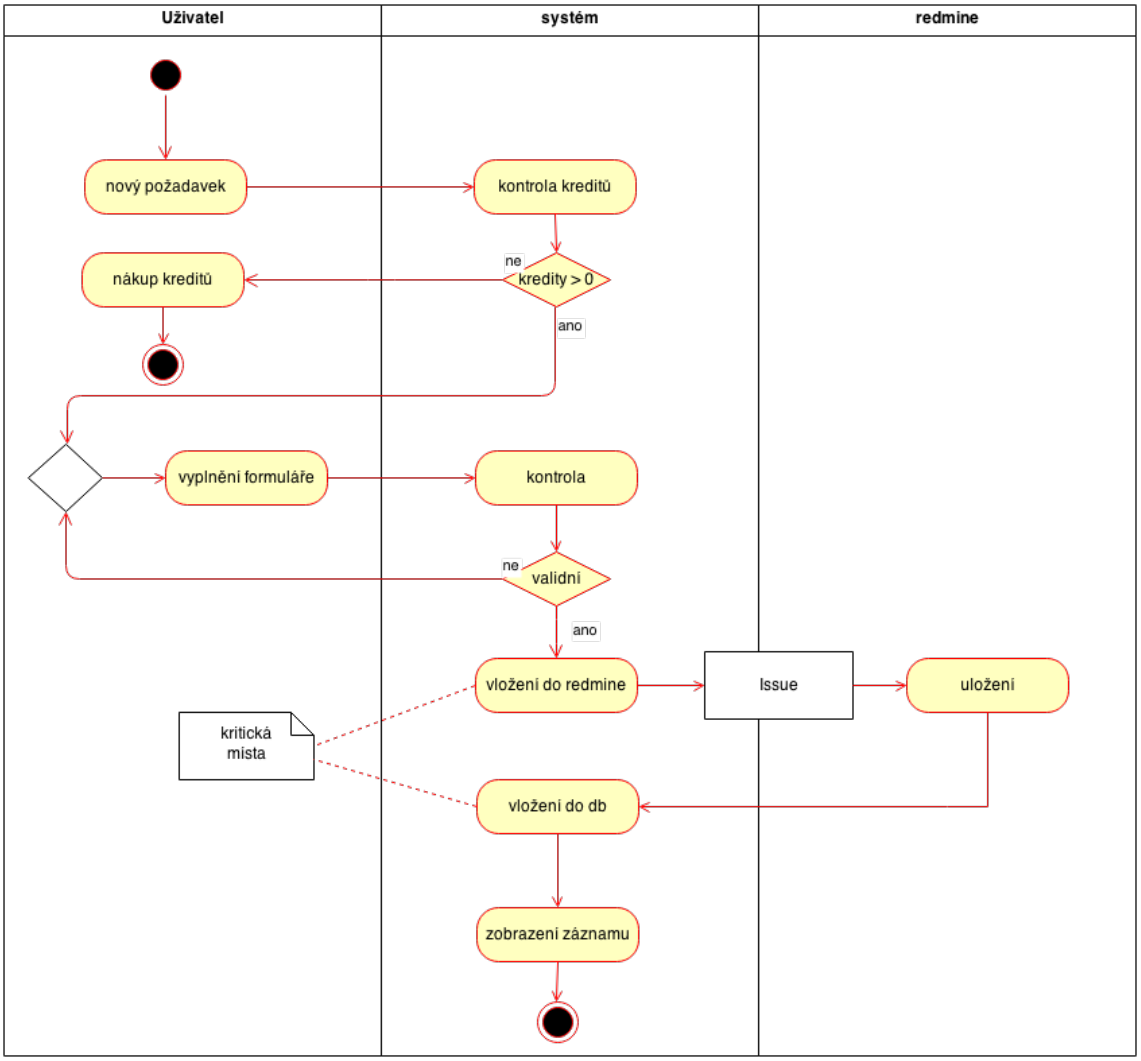
7.2 Podobná již existující řešení

Jelikož potřebujeme řešení, které sedí přesně naším specifickým požadavkům a systému práce, je velice těžké najít něco obdobného. RedmineCRM[28] poskytuje několik pluginů do Redmine. Aktuálně využíváme pluginy CRM a Helpdesk, které umožňují komunikovat se zákazníkem přes Redmine skrz email. Neobsahuje však žádné grafické rozhraní pro klienty, se kterým by se dalo dále nějak pracovat. Tyto pluginy slouží skvěle jako doplněk stávajícího systému, ale samy o sobě nesplňují naše požadavky.

[28] `http://www.redminecrm.com/`

7.3 Analýza

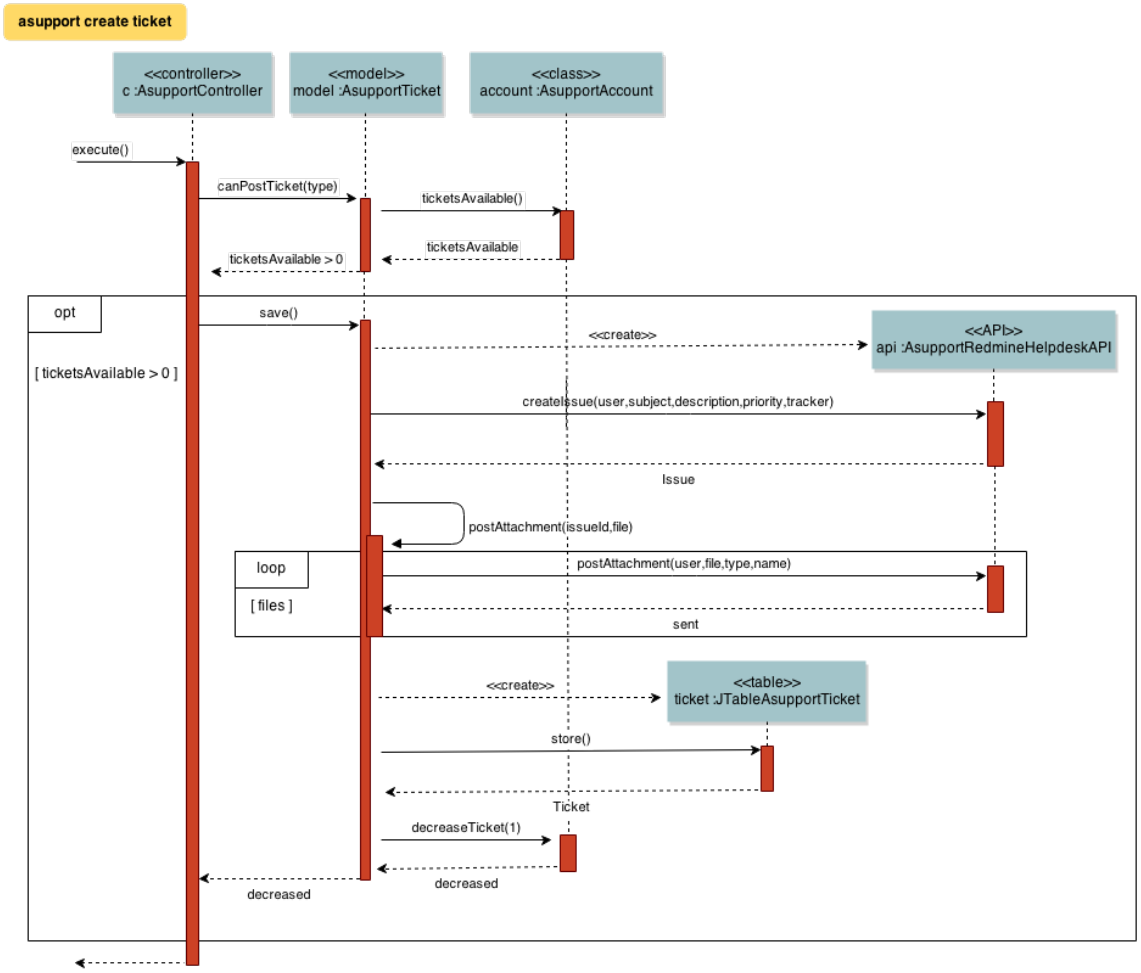
Diagram aktivit zobrazuje nedůležitější část systému a tou je Vložení nové podpory (specializace Vložení nového ticketu). Jako první důležitý krok je kontrola, zda má zákazník dostatek kreditů k vytvoření požadavku. V dalším kroku uživatel vyplní formulář, který se validuje a poté se přechází ke kritické části kódu. Systém vytvoří přes Redmine API nové issue s informacemi z formuláře a poté uloží zjednodušený záznam do databáze. Poslední finální krok je zobrazení požadavku.



Obrázek 3: Diagram aktivit - Vložení nové podpory

7.4 Návrh řešení

Tento sekvenční diagram zobrazuje jednu z nejdůležitějších operací a tou je vytvoření nového požadavku. Komponenta využívá MVC architekturu a proto jsou v diagramu patrné třídy pro Controller a Model. View není pro přehlednost v diagramu znázorněno. Nacházelo by se na konci diagramu. Třída Ticket využívá návrhového vzoru Active Record a přímo tak ukládá data do databáze. O připojení a komunikaci s REST Redmine API se stará třída Redmine, která v tomto případě vytváří nový záznam (issue) na vzdáleném serveru.



Obrázek 4: Sekvenční diagram - Vložení nového ticketu

Pro přípojení k Redmine API je využívána jíž existující knihovna, kterou je potřeba obohatit o specifickou funkcionalitu. AbstractApi je hlavní třída knihovny ze které dědí

všechny operace, které je možné nad API provádět. Rozšíření Ticket je vlastní implementace třídy, která vytváří nový ticket přes API Helpdesk pluginu Redmine.

AsuportApi je obecná třída pro načtení konkrétního API, která je využívána komponentou. AsupportRedmineApi pak konkrétně implementuje funkčnost, jež komunikuje s API vzdáleného serveru skrz RedmineApi knihovnu. AsupportRedmineHelpdeskApi je speciální rozšíření, využité pro práci s Helpdesk pluginem v Redmine.

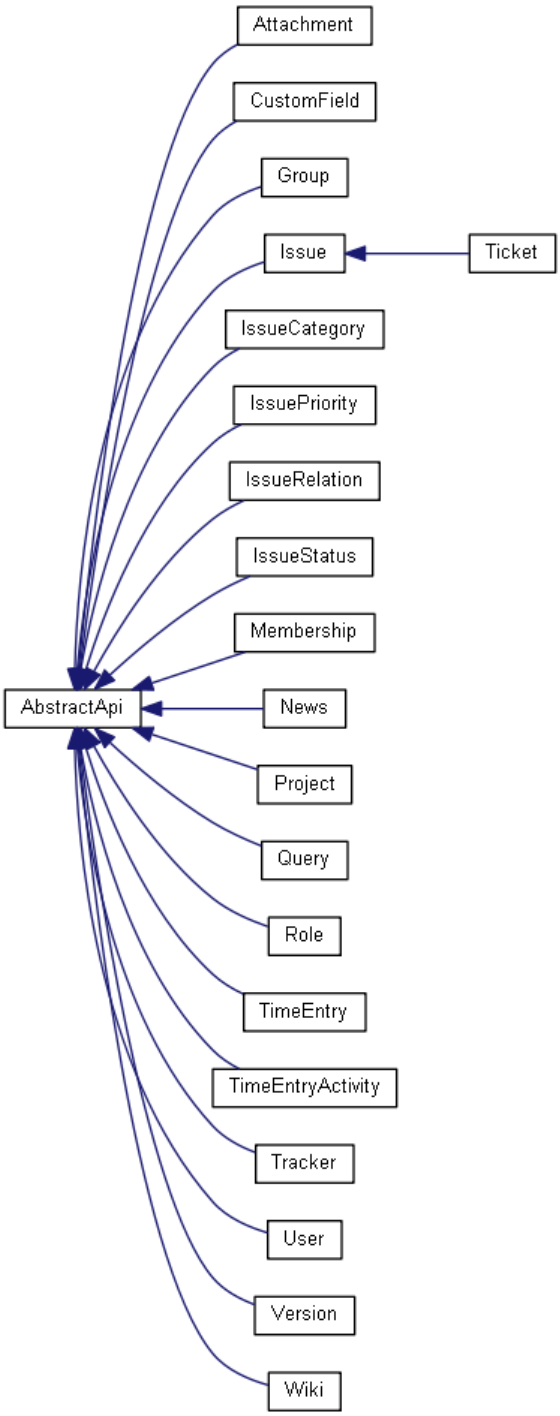
Aplikace využívá architektury MVC, proto jsou zde uvedeny třídy pro Controller, Model View. JControllerLegacy je abstraktní třída Joomla. Její potomek AsupportController je hlavní kontrolér komponenty, avšak pro speciální funkcionalitu je tento kontrolér ještě rozšířen. Kontrolér je vstupní bránou do aplikace.

Model obsahuje metody pro práci s daty, převážně pak ukládání, filtrování a zobrazení záznamů. AsupportModelTicket je třída pro práci s jednotlivými issue. O zobrazení a filtrování požadavků se stará AsupportModelTickets, stejnou funcionalitu pro sdílené požadavky zprostředkovává AsupportModelStickets

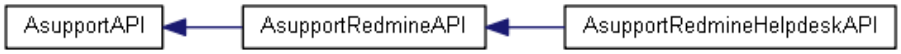
View třídy slouží k předávání dat z Modelu do šablony, která se stará o zobrazení. AsupportViewTicket slouží pro vytváření a zobrazení požadavku, AsupportViewTickets zobrazuje seznamy požadavků.

Pro mapování objektů na databázi slouží třída JTable. Její potomci obsahují konkrétní parametry (sloupce databáze) a metody (speciální funkcionalita). Již z názvu tříd je patrné, že v databázi jsou minimálně tabulky Account, Log, Participant, Ticket

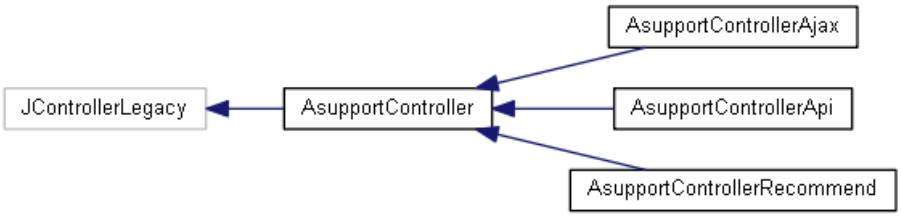
JPlugin je třída frameworku, plgSystemAsupport je konrétní implementace pluginu, který převádí zakoupené hodiny a tickety do uživatelského účtu.



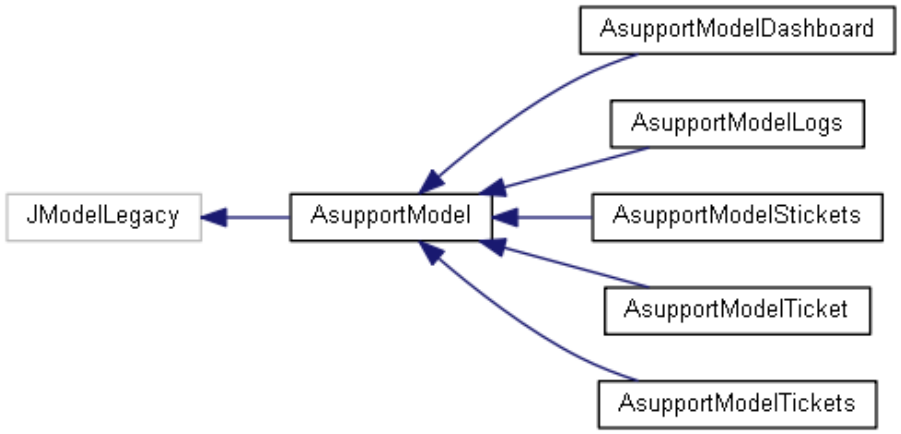
Obrázek 5: API knihovna



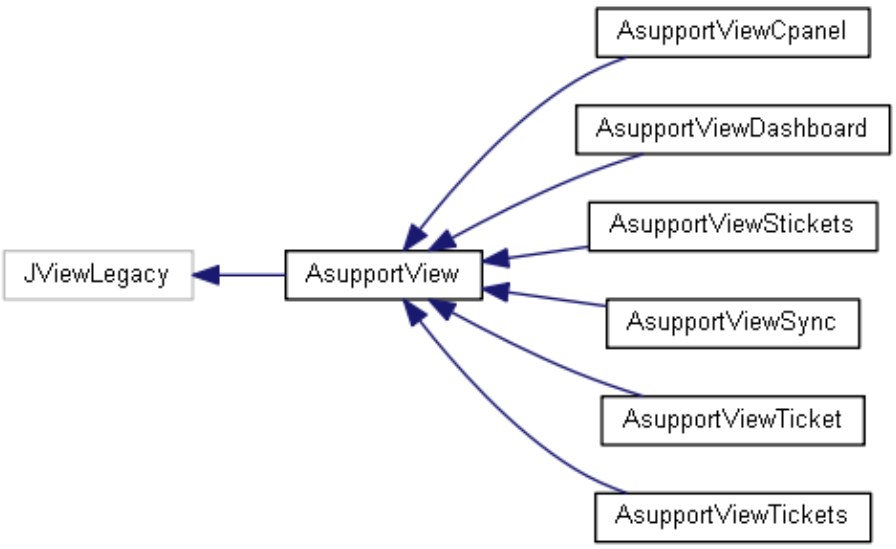
Obrázek 6: Specializace pro CRM API



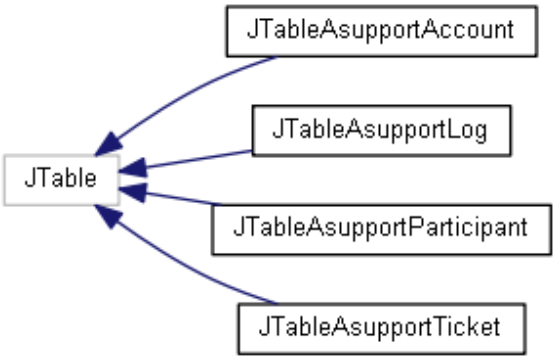
Obrázek 7: Kontrolér



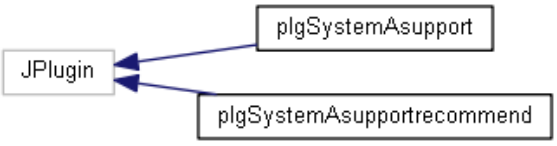
Obrázek 8: Model



Obrázek 9: Zobrazení



Obrázek 10: Active resource

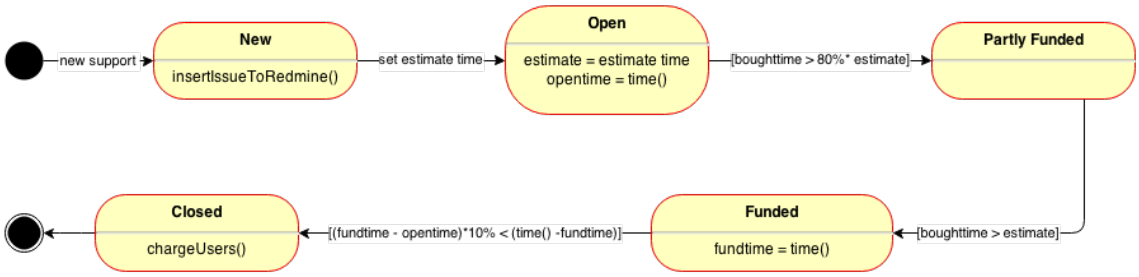


Obrázek 11: Plugin

7.5 Implementace

Stavový diagram

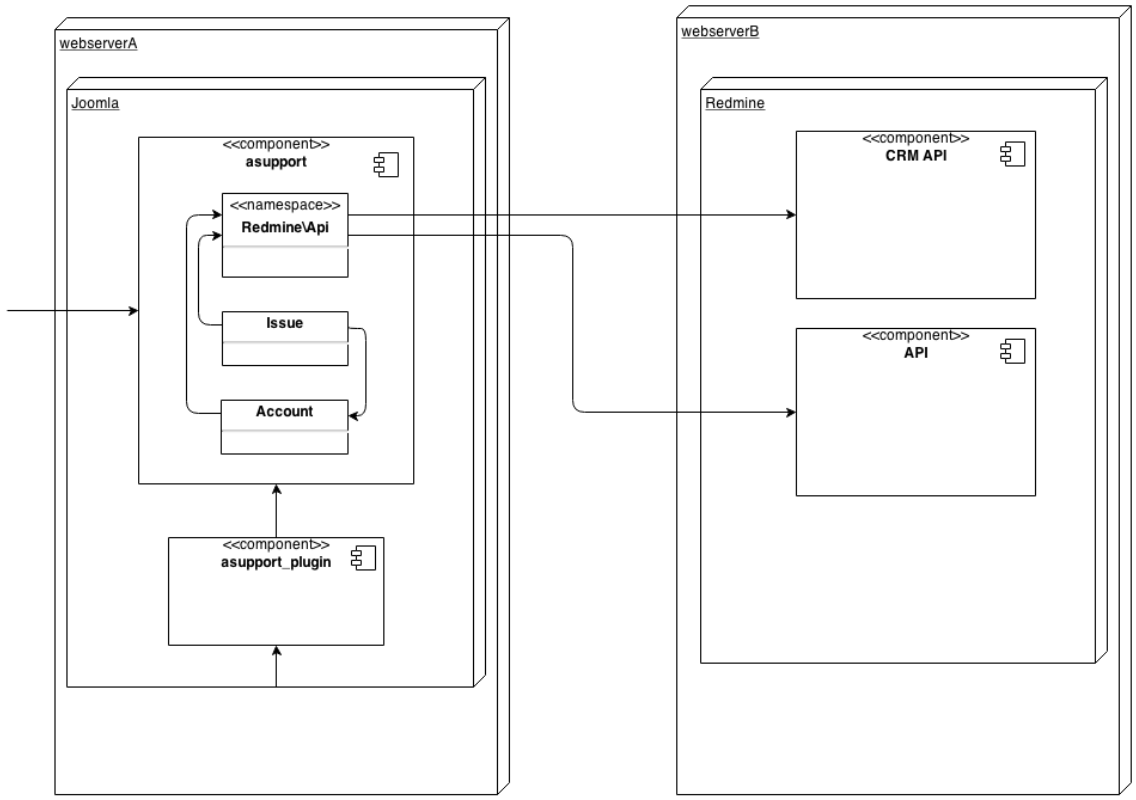
V tomto diagramu je znázorněna postupná změna vnitřních stavů sdíleného požadavku v komponentě. Nově vytvořený požadavek má stav New, pokud se v Redmine stanoví odhadovaná časová náročnost, požadavek se přepne do stavu Open. V tomto stavu je již viditelný pro ostatní zákazníky a ti s k němu mohou připojit. Pokud součet "nakoupených"hodin od všech uživatelů dosáhne 80% odhadované časové náročnosti, je požadavek přepnut do stavy Partly Funded, což znamená, že se již přestane zákazníkům zobrazovat procento "zaplacených"hodin. Pokud dojde k plnému zaplacení požadavku, požadavek se přepne do stavu Funded, kde setrvá po určitý čas a potom se automaticky uzavře do stavu Closed. Ve stavu closed dojde k úhradě hodin, k požadavku se již není možné připojit a začíná práce na implementaci. Skrytí zobrazení průběhu v posledních 20% a setrvání ve stavu Funded má psychologický význam, aby uživatelé byli motivováni k "nákupu"hodin, jelikož si nemohou být jisti, že funkčnost je plně uhrazena.



Obrázek 12: Stavový diagram - stav veřejného (sdíleného) požadavku

Diagram nasazení

Na diagramu je znázorněno reálné nasazení systému. Sytém se skládá z jednoduchého Joomla pluginu, který obstarává převod zakoupených hodin/ticketů z externí komponenty VirtueMart do hlavní komponenty pro podporu. Tato komponenta je tvořená třemi prvky. Klíčová je práce s uživatelským účtem (Account) a s požadavky (issue). API knihovna pak slouží jako brána pro komunikaci s Redmine. Redmine obsahuje komerční plugin Helpdesk (CRM) pro řízení vztahů se zákazníky, která funkčně rozšiřuje základní issues v Redmine a má rovněž své API, pomocí kterého vytváříme nové issue s možnosti přiřazení konkrétního zákazníka.



Obrázek 13: Diagram nasazení

7.5.1 API

Komunikace s Redmine je poměrně jednoduchá díky správně zvolené API knihovně. Nejdůležitější funkcí je vkládání nových issue do Redmine a načítání těchto issue. Další používané funkce jsou vkládání komentářů, upload souborů a změna issue. Trošku obtížnější je vytváření nového ticketu přes Helpdesk API. Zde bylo nutné rozšířit funkčnost knihovny a to hned ve třech třídách (Ticket, AsupportRedmineHelpdeskAPI, ClientHelpdesk).

Při každé operaci s Redmine jsou odchyceny případné výjimky (Exception), poté jsou chybové zprávy jsou uloženy do log souboru a zároveň odeslány emailem administrátorovi. Tímto přístupem je zajištěno okamžité informování o případné chybě. Méně příjemnou vlastností je, že Redmine API může korektně vrátit chybovou zprávu. V tomto případě nedojde k vyvolání výjimky a tato chyba musí být korektně ošetřena v kódu. SOAP protokol má v těchto případech standardizované posílání chyb, ale REST nic takového nemá a chybové zprávy se liší implementaci od implementace.

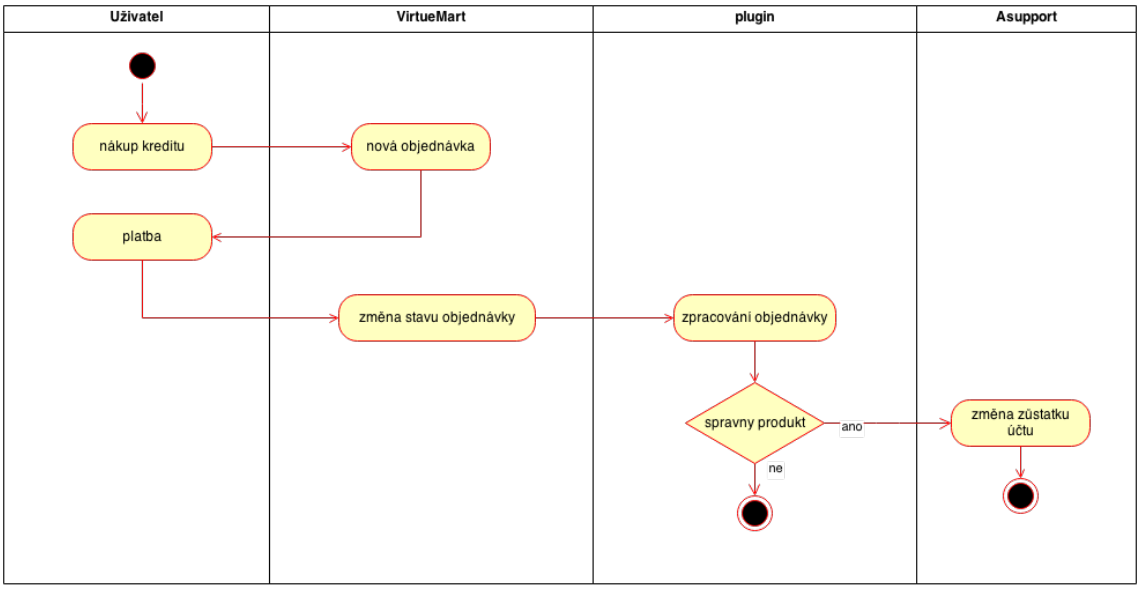
```
$this->client = new Redmine\ClientHelpdesk($config->apiUrl, $config->apiKey);
try{
    $data = $this->client->api('ticket')->create($issue, $user);
}catch(Exception $e){
    $this->setApiErrorMsg($e->getMessage());
}
```

Výpis 8: Vytvoření nového ticketu (issue)

7.5.2 Platby

Nové hodiny lze zakoupit přes přímý odkaz vedoucí k položce v komponentě internetového obchodu VirtueMart[29]. Po zakoupení se objednávka uloží do databáze a po zaplacení se změní její stav. Při změně objednávky se spustí plugin komponenty, kterému se předá objednávka. Plugin zpracuje objednávku a změní zůstatek na uživatelském účtu zákazníka.

[29] http://www.virtuemart.net/



Obrázek 14: Diagram aktivit - nákup kreditů

8 Analytický systém

Tato kapitola, stejně jako předchozí, se pokusí popsat návrh a realizaci řešení analytického systému. Postup je velice podobný a proto se i zde bude vyskytovat zápis požadavků pomocí metody FURPS. Případy užití jsou jen zkráceně vyjmenovány, neboť se zde nejedná o žádnou rozsáhlou funkcionalitu. Mnohem důležitější jsou zde sekvenční a třídní diagramy, které poskytují ucelený přehled o fungování systému.

8.1 Požadavky

Zde probíhá podobný postup jako u komponenty Asupport7. Není však nutné detailně rozepisovat případy užití a proto jsou zde pouze vyjmenovány základní požadavky na případy užití dle metodiky Scrum[30]

8.1.1 Požadavky FURPS+

Funkčnost

1. systém umožní načíst požadavky z Redmine a provést jejich analýzu
2. systém umožní zobrazit pro každého pracovníka podíl času stráveného na jednotlivých projektech
3. systém umožní zobrazit pro každý projekt podíl času stráveného jednotlivými pracovníky
4. systém umožní zobrazit issue, které budou opožděny s realizací
5. systém umožní zobrazit předpokládané datum dokončení všech úkolů pro pro každého pracovníka
6. systém umožní odhadnout (predikovat), které issues budou pravděpodobně realizovány déle, než je předpokládáno

Použitelnost

1. Systém bude zobrazovat většinu dat pomocí grafů, případně přehledných tabulek.
2. Časté znovu načtení stránky se bude minimalizovat použitím vyskakovacích oken.
3. Webové rozhraní bude alespoň částečně responzivní.

Spolehlivost

1. Systém bude mít řádně ošetřené výjimky v kódu.

[30] `http://www.agilemodeling.com/artifacts/userStory.htm`

2. Systém bude správně vyhodnocovat problémy vzniklé při synchronizaci s Redmine.

Výkon

1. Synchronizace dat nebo jiné obdobné procesy nebudou prováděny v rámci komunikace uživatele se systémem, ale pomocí CRON služby.

Podporovatelnost

1. Zdrojový kód bude řádně okomentován.
2. Uživatel bude informován v případě chyby.

Návrhové požadavky
Systém může ukládat data libovolným způsobem. Preferovaný způsob je serializovaný JSON v souboru, relační nebo NoSQL databáze.

Implementační požadavky
Systém musí být napsán v programovacím jazyku PHP. Musí být použity pouze takové technologie, které je možné využít, případně snadno doinstalovat na běžném LAMP/WAMP serveru.

Požadavky na rozhraní
Komponenta musí být schopna přenést data z Redmine API.

Fyzické požadavky
Systém musí být spustitelný na běžné osobním počítači. Všechny výpočty musejí být realizovány v operační paměti počítače.

8.1.2 Základní požadavky na případy užití

Analytický systém má zobrazit manažerovi všechny potřebné údaje o aktuální vytíženosti pracovního týmu a statistické údaje ohledně úspěšnosti a náročnosti jednotlivých projektů. Manažer musí být schopen zjistit strávený čas pracovníka na projektu vzhledem k ostatním pracovníkům, stejně tak jako čas strávené na jednotlivých projektech. Dále pak je kritické vidět optimálně naplánované (seřazené) úkoly v přehledném Gantt diagramu. Jedním z nejdůležitějších ukazatelů je extra čas, který nám říká o kolik byl úkol prodloužen (nebo urychlen) oproti odhadu.

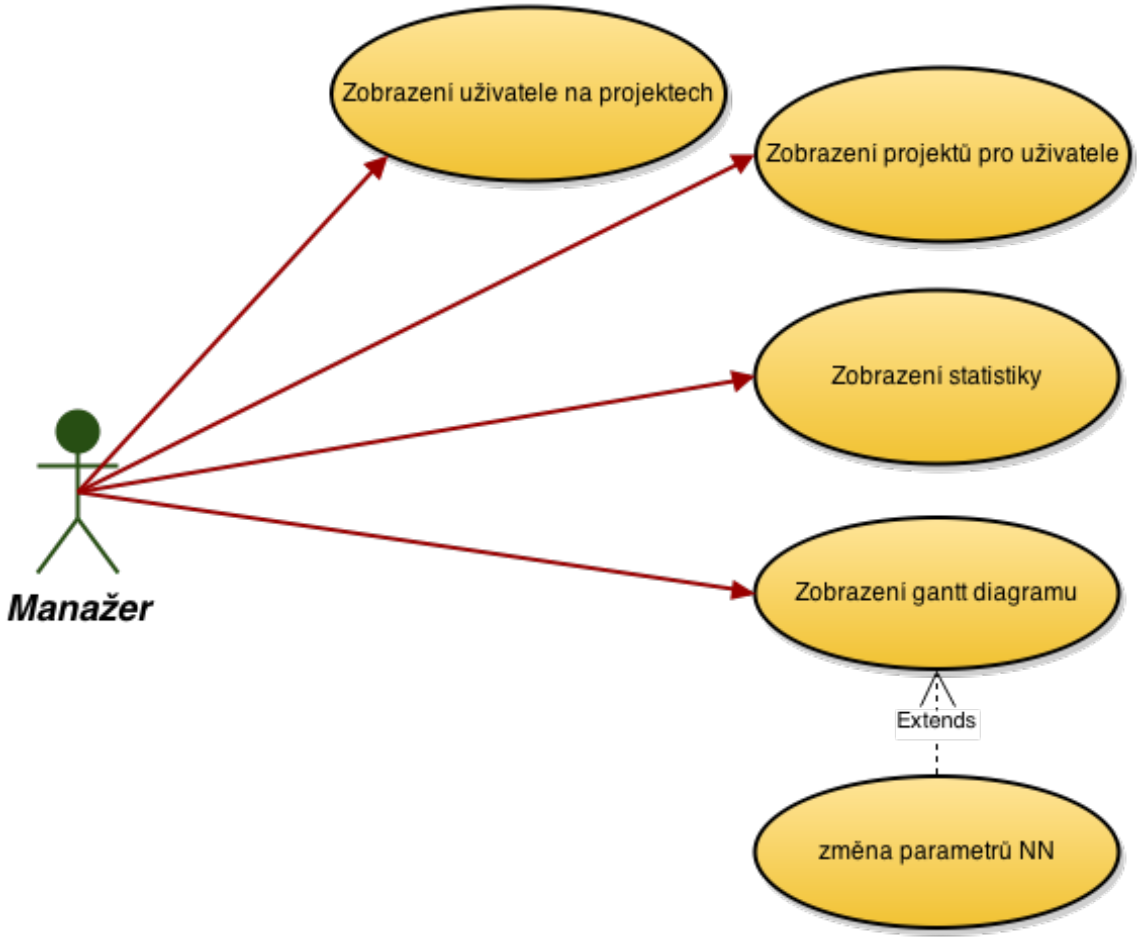
Základní požadavky na případy užití

- Jako manažer chci vidět kolik času poměrově tráví pracovníci na jednotlivých projektech.
- Jako manažer chci vidět poměr stráveného času vzhledem k ostatním pracovníkům na projektu.

- Jako manažer chci znát extra čas strávený na projektech vyjádřený procentuálně i hodinami.
- Jako manažer chci znát extra strávený čas pro každého pracovníka vyjádřený procentuálně i hodinami.
- Jako manažer chci vědět do kdy mají pracovníci naplánované úkoly.
- Jako manažer chci znát nejlepší možné řazení úkolů tak aby se neopozdily.
- Jako manažer chci vědět které úkoly jsou již opožděné nebo je nemožné je splnit do stanoveného data dokončení.
- Jako manažer chci vědět které úkoly se pravděpodobně opozdí s realizací vzhledem ke stanovenému datu dokončení.

8.1.3 Diagram případů užití

Tento diagram případů užití zobrazuje všechny možnosti, které se naskytují zákazníkovi. Případů užití je zde poměrně málo a nejsou nikterak komplexní. Případ užití Zobrazení Gantt diagramu je nejsofistikovanější neboť zahrnuje generování Gantt diagramu a práci s neuronovou sítí.



Obrázek 15: Diagram případů užití

8.2 Podobná již existující řešení

Na základě běžného průzkumu se lze domnívat, že v současné době neexistuje snadno dohledatelné řešení, které by přehledně a v grafech zobrazilo požadované informace.

Graphs Plugin je plugin[31], který v grafech zobrazí otevřené a zavřené issue, délku otevření issue, a počet opožděných issue. Tyto grafy se zdají být užitečné jako doplněk k tabulkovým informacím, ale nelze na jejich základě dělat strategické rozhodnutí nebo získat přehled o úspěšnosti projektů.

Easy Redmine je přepracovaná[32] verze Redmine, která je uživatelsky mnohem přívětivější, ale funkcionálně neobsahuje téměř nic navíc. Je možné dokoupit nejrůznější pluginy, mimo jiné i plugin Resource Management[33], který slouží k přiřazení uživatelů k issue v Gantt diagramu. Tento plugin podává přehled o vytíženosti uživatelů, ale bohužel toho víc neumí.

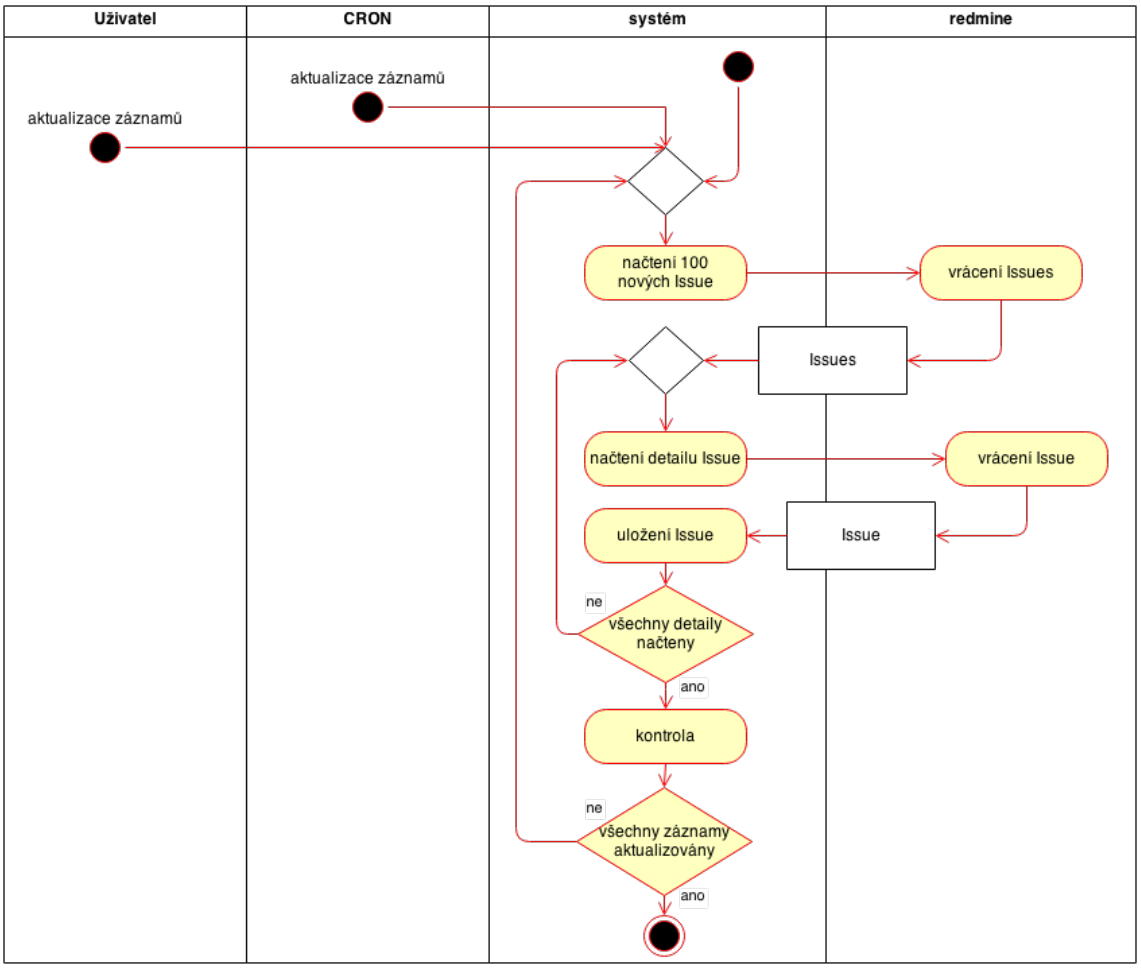
[31] http://www.redmine.org/projects/redmine/wiki/PluginGraphs

[32] http://www.easyredmine.com/online-store/list-all-products

[33] http://www.easyredmine.com/online-store/easy-redmine-plugins/resource-management

8.3 Analýza

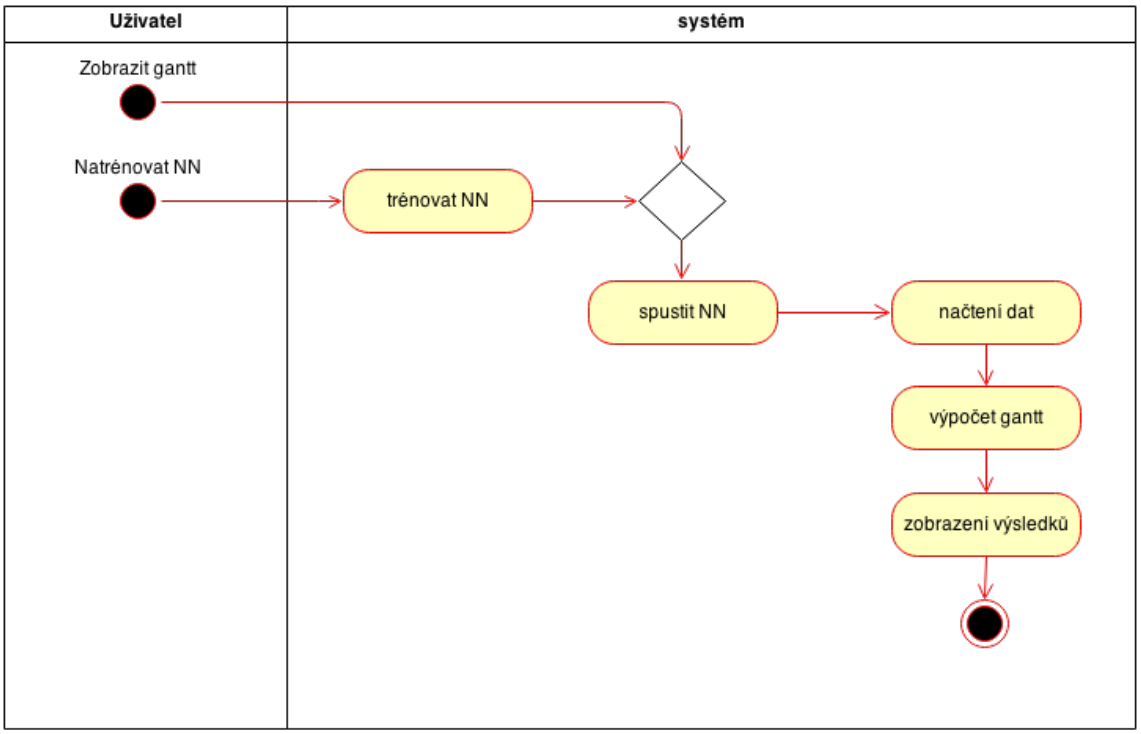
Diagram zobrazuje jednoduchý a vcelku logický postup získání dat pro analýzu. Klíčová je zde komunikaci s Redmine, jelikož je načítání dat ve smyčce. Při velkém objemu nových nebo upravených záznamů není možné tuto aktualizaci provádět během práce uživatele. Proto je nutné spouštět načtení dat pokud možno pomocí služby CRON. Zároveň je důležité hlídat dobu běhu scriptu tak, aby zbytečně nedocházelo k překročení časového limitu.



Obrázek 16: Diagram aktivit - Načtení dat

Zobrazení Gantt grafu, jež popisuje diagram aktivit, je nejsofistikovanější operací v analytickém systému a to zejména díky práci s neuronovou sítí. Pokud síť není natrénovaná, je potřeba vytvořit trénovací množinu, data normalizovat a natrénovat síť. Při každém zobrazení diagramu se pak díky neuronové sítě zobrazí úkoly, které budou

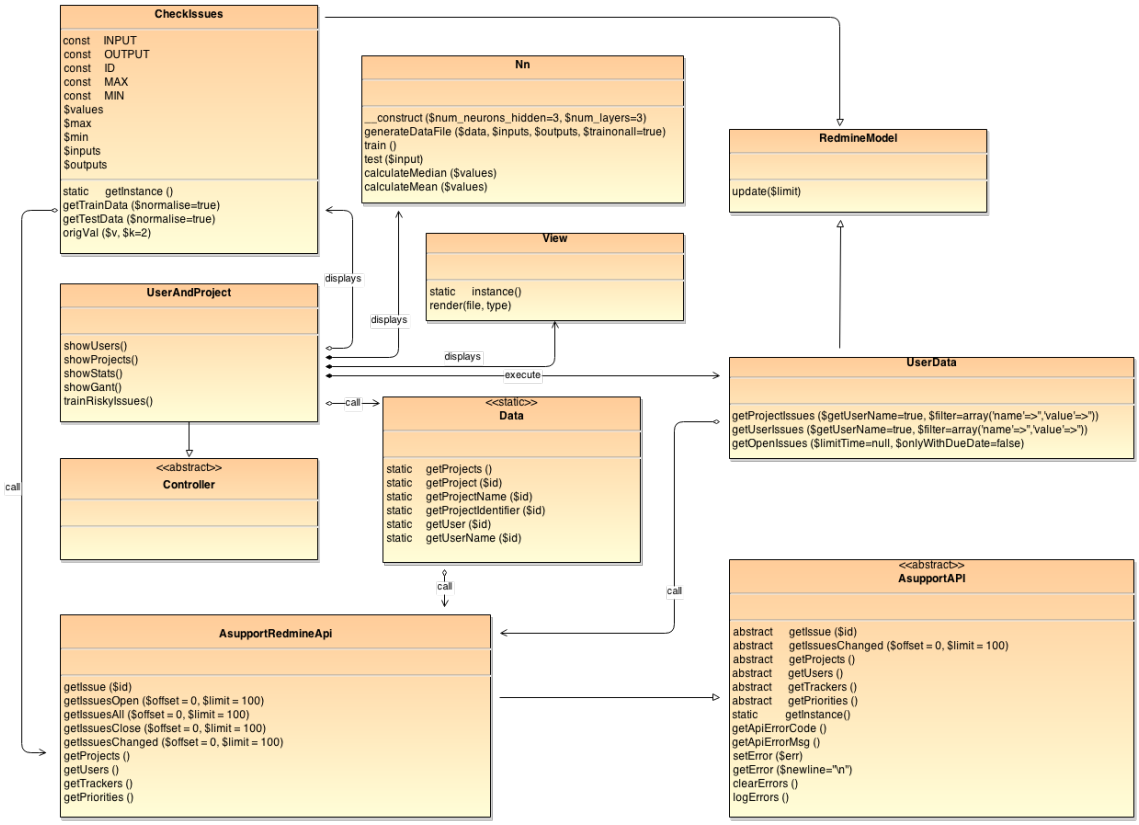
pravděpodobně opožděny. Výpočet Ganttova diagramu je druhou náročnější operací při tomto požadavku.



Obrázek 17: Diagram aktivit - Zobrazení Gantt diagramu

8.4 Návrh řešení

Tento diagram tříd popisuje celou aplikaci vsazenou do frameworku FatFreeFramework (f3). Hlavní třída je UserAndProject a obstarává výstup dat pro každý dotaz. Možné dotazy přímo odpovídají metodám této třídy. Třídy CheckIssues a UserData jsou návrhem velice podobné, avšak implementací jednotlivých metod se liší. Data poskytnuté třídou CheckIssues se využívají pro neuronovou síť. Třída Nn poskytuje rozhraní pro práci s neuronovou sítí FANN (Fast Artificial Neural Network[34]). Statická třída Data slouží pouze pro uložení používaných názvů z Redmine tak, aby nedocházelo k neustálým dotazům přes API. AsupportAPI je identická třída se třídou, která se používá v komponentě Asupport ke komunikaci s Redmine API.

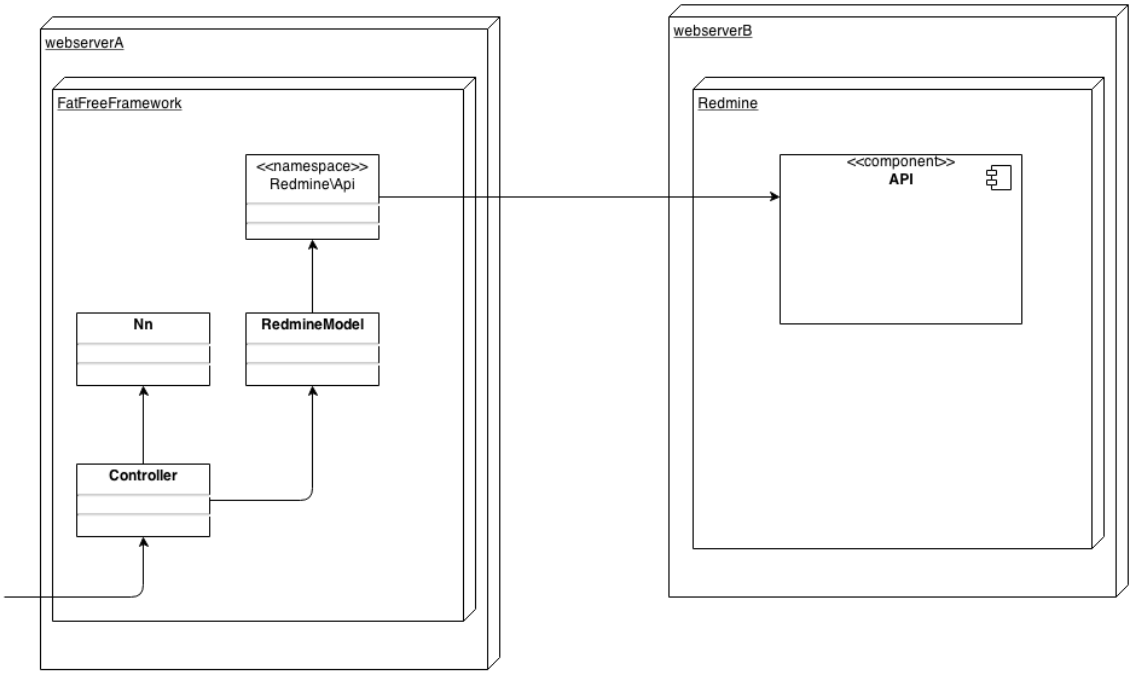


Obrázek 18: Třídní diagram

[34] http://leenissen.dk/fann/wp/

8.5 Implementace

Diagram nasazeni zobrazuje celky systému, jejich propojení a závislosti. Analytický systém je implementován v rámci malého frameworku FatFreeFramework (f3). Skládá se ze čtyř hlavních částí. Klíčovou částí je RedmineModel, kde probíhá práce a výpočty s daty. Controller je centrální prvek, který obstarává jednotlivé akce a taktéž částečně generování HTML kódu. Nn je jednoduchá třída pro práci s Neuronovu sítí (FANN).



Obrázek 19: Diagram nasazení

8.5.1 Strojové učení

Strojové učení, konkrétně pak neuronové sítě, bylo experimentálně využito pro predikci issues, které se budou příliš odchylovat od ostatních issue. V tomto případě to znamená najít taková issue, která budou pravděpodobně v budoucnosti vyžadovat více času než bylo předpokládáno. Tyto issue nazýváme riskantní, protože je u nich predikována větší časová, tedy i finanční náročnost než je stanoveno/odhadováno. Z API Redmine máme dostatečné množství informací uložených v databázi a proto tyto data můžeme analyzovat.

Predikovat riskantní issue by se dalo přirovnat k věštění z magické koule. Je potřeba najít takové parametry, které by výstižně popisovaly aktuální stav issue. Během procesu bylo nutné vyzkoušen mnoho kombinací a postupů. Většina pokusů vedlo k neuspokojivým výsledkům. Prvotní vize predikovat riskantní issue se změnila v dlouhý a náročný

proces, s nejistým výsledkem. Nakonec se díky neobvykle zvolenému postupu podařilo dojít k částečné relevantním výsledkům.

Neuronová síť

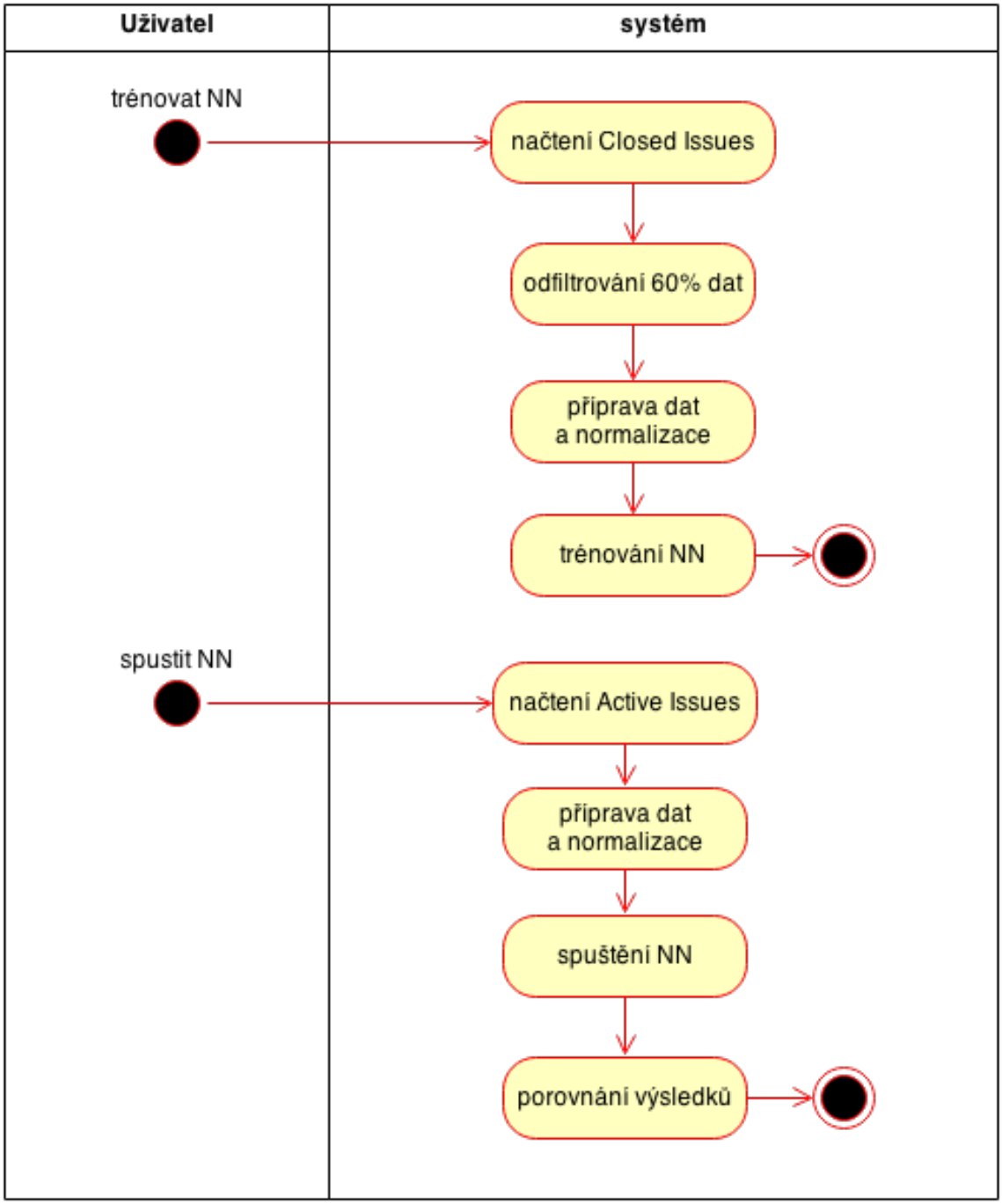
Ačkoliv byla práce na neuronové sítí časové velice náročná, popíšu zde vše jen stručně, neboť toto není hlavní náplní této práce. Práce s neuronovou sítí se skládá ze dvou kroků. Jako první je třeba získat vstupní data i s požadovaným výstupem, tyto data se nazývají trénovací set. Neuronová síť se při trénování (učení) dostane do takového stavu, kdy pro dané vstupní data vrátí požadovaný výstup. Poté následuje fáze spuštění neuronové sítě, kdy máme data u kterých neznáme výstup a naučená neuronová síť vrátí (pravděpodobně) správný výstup.

Učení neuronové sítě

Jako vstupní hodnoty jsou číslo projektu a odhadovaný čas potřebný k dokončení issue. Výstupní hodnoty jsou délka popisu problému a počet změn issue. První je třeba vytvořit trénovací množinu. Načteme tedy všechny issue, které jsou ukončené a mají stanovený časový odhad. Pro všechny tyto issue se vypočítá o kolik extra procent bylo na issue stráveno více nebo méně času. Poté se seřadí issue podle extra procent vzestupně a vybere se horních 30%. Tento výběr představuje issue, kde strávený čas byl nejblíže odhadovanému. Tímto užším výběrem se natrénuje neuronová síť a ta se uloží do souboru pro další použití.

Spuštění neuronové sítě

V případě, že chceme stanovit odhad riskantních issue, vybereme z databáze stále otevřené (aktivní) issue se stanoveným časovým odhadem, načteme dříve vytvořenou neuronovou síť a získáme vypočítané hodnoty na základě vstupu otevřených issue. Neuronová sít nám tedy řekne, jak dlouhý by měl být popisek "dobrého"issue a kolikrát by měl změnit stav. Tyto vypočítané údaje porovnáme se skutečnými hodnotami pro dané issue a dostaneme chybu. Čím větší je chyba, tím pravděpodobnější je, že dané issue nespadá do kategorie 30% nejlepších issue. Poté už jenom stačí zobrazit issue s největšími chybami, které představují riskantní issue.



Obrázek 20: Diagram aktivit neuronové sítě

9 Vyhodnocení

V této závěrečné kapitole jsou popsány obtížné části celé práce a také to, jak bude celý projekt hodnocen. Při vyhodnocení je brán v potaz převážně reálný přínos implementovaného systému.

9.1 Poznámky k řešení

Zde jsou popsány největší problémy a obtíže, které se naskytly během vývoje jednotlivých řešení.

Asupport
Nejobtížnější bylo stanovení byznys procesů a pravidel. Tento systém je navržen jako komplexní nástroj s vysokou mírou automatizace. Proto bylo důležité přesně zmapovat procesy a při implementaci myslet doslova na každou drobnost. Čím vyšší stupeň automatizace, tím obtížnější je provést ruční zásah v případě chyby. Kvůli Helpdesk pluginu v Redmine bylo potřeba poupravit práci s API a proto zobrazení zpráv u požadavku není z pohledu zákazníka úplně ideální.

Analytický systém
Volba vhodného způsobu uložení dat byla dlouhou dobu otevřená otázka. První verze fungovala pouze s ukládáním dat do souboru, ale to bylo z dlouhodobého hlediska neudržitelné (nevhodné). Nakonec jsem zvolil MySQL databázi z důvodu jednoduchého nasazení do provozu. Ačkoliv zprvu vypadala MongoDB databáze jako lepší volba, tento projekt by nevyužil žádnou z unikátních vlastností NoSQL a proto neexistuje vážný důvod k tomuto nasazení. Daleko nejobtížnější byla predikce riskantních issue. Díky nesourodosti dat byl velký problém odhalit parametry a vlastnosti, jenž mají požadovaný vliv. Nakonec došlo k poměrně neobvyklému řešení, které funguje překvapivě dobře, avšak stále je potřeba pamatovat, že se jedná pouze o doporučení na základě predikce, nikoliv přesné klasifikování. Najít vhodnou neuronovou síť pro PHP bylo velice obtížné, protože mnohé PHP knihovny nefungovaly podle představ. Finální řešení využívá známou FANN knihovnu a to díky PEAR rozšíření[35] pro PHP.

9.2 Vyhodnocení

Jelikož se systém nasazuje do provozu právě v době odevzdávání této diplomové práce, není možné okamžitě zhodnotit jeho úspěšnost či neúspěšnost. První vyhodnocení je vhodné udělat po měsíci provozu, ale největší vypovídací hodnotu budou mít hodnocení dlouhodobého chodu systému. Proto je zde popsáno jak bude probíhat hodnocení systému po delším časovém období.

Vyhodnocení bude provedeno dvěma způsoby. První způsob přesně měří dosažených výsledků. Druhý způsob si zakládá na subjektivním, daty nepodloženým, hodnocením

[35] `http://pecl.php.net/package/fann`

od řídících pracovníků. Oba přístupy mají své opodstatnění a lze jen těžce určit, který z hodnocení má větší vypovídací hodnotu. V ideálním případě by měly předložené data korespondovat se subjektivním názorem managementu. Změřené vyhodnocení se zabývá hlavně finančními výsledky. Jednoduché propočty růstu poskytnou přesný přehled o finanční bilanci.

Subjektivní názor může být velice zavádějící, ale někdy také jako nejlepší způsob vyhodnocení. Určení celkového finančního dopadu dané změny dle předchozí metody je velice obtížné. Spokojení pracovníci a dobrá nálada v týmu se na finančních výsledcích společnosti může projevit až za poměrně dlouhou dobu, ale v osobních vztazích a spokojenosti v práci se zajisté projeví okamžitě. Pokud nový systém ulehčí práci vývojářům, ale nezlepší okamžitou finanční bilanci, splní částečně očekávání. Z těchto důvodů je nutné do celkového hodnocení zahrnout i subjektivní názory, které mohou mít větší skutečnou vypovídací hodnotu, než finanční bilance společnosti[36][37].

Změřené

Toto vyhodnocení se zabývá konkrétními finančními výsledky. Nejběžnější metodou jak zjistit růst společnosti je porovnání zisku v dvou navazujících obdobích, protože propočty růstu zisku se výrazně projevují na finanční bilanci společnosti.[15] Výsledkem tohoto porovnání růstu nebo ztráty je tempo růstu[38] (PR) v %. Zisk za aktuální období je označen jako A, zisk z předešlého období je označen jako P.

Vzorec pro výpočet tempa růstu

$PR = \frac{A-P}{P} * 100$

Příklad 9.1

Zisk v předešlém období $P = 1$ zisk v aktuálním období $A = 1,1$.

$PR = \frac{1,1-1}{1} * 100 = 10$

Tempo růstu neboli navýšení zisku za období A oproti období P je 10%. Tento údaj nám bohužel neříká, co může za zvýšení zisku a jak se na něm podepsaly změny provedené v aktuálním období (A). Pokud by byl růst globální ekonomiky mezi těmito obdobími 5%, pak nové řešení zvýšilo zisk potenciálně pouze o $10\% - 5\% = 5\%$. Pokud by ale byl růst konkrétně malých společností, zabývající se vývojem pro web, 15%, pak by nové řešení mohlo způsobit ztrátu $10\% - 15\% = -5\%$. ■

Jako další faktory vstupují četné vnitro firemní změny a strategická rozhodnutí. Například nábor nových špičkových programátorů zvedne náklady na lidské zdroje a krátkodobě "prodraží"vývoj. Po dostatečném zaškolení mohou tito zkušení vývojáři udat nový směr vývoje a podstatně vylepšit stávající produkt, což by mělo vést ke zvýšení prodeje v dalších kvartálech nebo letech. Přímý finanční dopad dané změny je možné spočítat jen velice obtížně a proto je vhodné zavést jiné ukazatele.

[36] http://www.penize.cz/ekonomika/262764-jak-merit-uspech

[37] http://www.bothsidesofthetable.com/2011/12/27/should-startups-focus-on-profitability-or-not/

[38] http://pages.uoregon.edu/rgp/PPPM613/class8a.htm

Vzhledem k Vizi5 je hlavním cílem minimalizovat dotazy a navýšit počet placených úprav komponent. Opět použijeme vzorec pro tempo růstu ale mírně pozměníme vstupní proměnné. Výsledek za aktuální období (A) je vyjádřen jako poměr všech dotazů zákazníku a placených úprav za období. Dotazy zákazníku za období A jsou označeny jako DA, placené úpravy za období A jsou označené jako UA.

Analogicky stejné označení platí i pro předešlé období (P). Dotazy od zákazníků jsou označeny jako DP a placené úpravy jako UP. $A = \frac{DA}{UA}$ $P = \frac{DP}{UP}$

Dosazením těchto proměnných do vzorce pro výpočet tempa růstu $PR = \frac{A-P}{P} * 100$ získáme nový, mírně modifikovaný vzorec. Výpočet tempa růstu placených úprav

$$PR = \frac{\frac{DA}{UA} - \frac{DP}{UP}}{\frac{DP}{UP}} * 100$$

Vypočtený zisk za první měsíc provozu nového systému se musí porovnat s výsledky za předešlý měsíc nebo za stejný měsíc minulého roku. Zde nastává problém při kterém nemůžeme s jistotou říci, zda podmínky na trhu byly v podobné. Pokud jsme schopni rozdíl na trhu doložit, je vhodné tyto rozdíly zohlednit ve výpočtu. Problém tohoto postupu je, že ačkoliv jsme empiricky provedly výpočty, správnost lze jen těžce ověřit a proto může dojít ke zkreslení výsledku. Z těchto není vhodné tuto metodu použít pro detailní vyhodnocení. Růst společnosti je detailně popsán v [16, 17, 18].

Subjektivní

Ve větších organizacích by bylo vhodné vytvořit elektronické dotazníky pro zpětnou vazbu, které by se rozeslaly všem zúčastněným a subjektivní hodnocení by se provádělo na základě odpovědí respondentů. V našem případě jsou dotazníky naprosto zbytečné. Postačí se osobně sejít s každým interním pracovníkem a krátce s ním pohovořit o přínosu systému, přičemž odpovědi řídících pracovníků a manažerů mají samozřejmě největší váhu.

10 Závěr

V rámci této diplomové práce jsem vytvořil dva nezávislé systémy. První systém řeší komplexně a vysoce automatizovaně proces zadávání požadavků. Klient může vést veškerou komunikaci přes tuto platformu, stejně tak jako provádět platby za úpravy. Systém umožňuje sdílené financování veřejného požadavku, což je unikátní vlastnost, která je přínosná pro zákazníky i pro společnost ARTIO.

Analytický systém obsahuje nástroje, které zatím nebyly pro Redmine dostupné. Kromě statisticky zajímavých informací obsahuje i seznam predikovaných riskantních issue, které pravděpodobně zaberou více času než stanovený odhad. Nejhodnotnější je funkce Ganttova diagramu, který navrhne nejvhodnější sekvenční plnění nových úkolů pro každého pracovníka. Toto minimalizuje opoždění úkolů, zlepší přehled o úkolech a tím i sníží stres pracovníkům.

10.1 Budoucí práce

Komponenta Asupport v současné době splňuje veškeré požadavky a proto se zde nepředpokládá rozsáhlejší vývoj. Při běhu systému budou pravděpodobně provedeny pouze drobné funkcionální změny. Analytický systém svou funkčností naprosto splňuje stávající potřeby managementu. Rozšíření připadá v úvahu pouze v případě, že najde uplatnění v každodenním chodu firmy. Za těchto okolností by bylo vhodné zvážit, zda nenabízet tuto platformu formou SaaS společnostem využívajícím Redmine. V tomto případě by došlo k podstatnému rozvoji tohoto analytického systému s následným, plně oprávněným, přechodem na NoSQL databázi.

11 Reference

[1] LIM, John. *Object-Relational Mania.* PHP Everywhere [online]. 20.6.2004 [cit. 2014-04-29]. Dostupné z: http://phplens.com/phpeverywhere/node/view/25

[2] FOWLER, Martin. *OrmHate.* Martin Fowler [online]. 8.5.2012 [cit. 2014-04-29]. Dostupné z: http://martinfowler.com/bliki/OrmHate.html

[3] *The Object-Relational Impedance Mismatch.* Agile Data [online]. c 2002-2013 [cit. 2014-04-29]. Dostupné z: http://www.agiledata.org/essays/impedanceMismatch.html

[4] BERENSON, Hal, Phil BERNSTEIN, Jim GRAY, Jim MELTON, Elizabeth O'NEIL a Patrick O'NEIL. *A Critique of ANSI SQL Isolation Levels.* In: A Critique of ANSI SQL Isolation Levels [online]. 1995 [cit. 2014-04-29]. Dostupné z: http://research.microsoft.com/pubs/69541/tr-95-51.pdf

[5] CHANG, Fay, Jeffrey DEAN, Sanjay GHEMAWAT, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes a Robert E. Gruber. *Bigtable: A Distributed Storage System for Structured Data.* In: Bigtable: A Distributed Storage System for Structured Data [online]. 2006 [cit. 2014-04-29]. Dostupné z: http://goo.gl/jeayl

[6] *Encyclopedia Index.* DB-Engines [online]. c 2012-2014 [cit. 2014-04-29]. Dostupné z: http://db-engines.com/en/articles

[7] GILBERT, Seth a Nancy LYNCH. *Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services.* In: Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services [online]. 2002 [cit. 2014-04-29]. Dostupné z: http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf

[8] PRITCHETT, Dan. *The Challenges of Latency.* Facilitating the spread of knowledge and innovation in professional software development [online]. 2.5.2007 [cit. 2014-04-29]. Dostupné z: http://www.infoq.com/articles/pritchett-latency

[9] YU, Haifeng a Amin VAHDAT. *Design and Evaluation of a Continuous Consistency Model for Replicated Services.* In: Design and Evaluation of a Continuous Consistency Model for Replicated Services [online]. 2000 [cit. 2014-04-29]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.34.7743&rep=rep1&type=pdf

[10] ROE, Charles. *ACID vs. BASE: The Shifting pH of Database Transaction Processing.* DATAVERSITY [online]. 1.4.2012 [cit. 2014-04-29]. Dostupné z: http://www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing/

[11] *Why NoSQL?: Three trends disrupting the database status quo.* Couchbase [online]. c 2014 [cit. 2014-04-29]. Dostupné z: http://www.couchbase.com/why-nosql/nosql-database

[12] HURST, Nathan. *Visual Guide to NoSQL Systems.* Nathan Hurst [online]. 15.3.2010 [cit. 2014-04-29]. Dostupné z: http://blog.nahurst.com/visual-guide-to-nosql-systems

[13] BRUST, Andrew. *RDBMS vs. NoSQL: How do you pick?* ZDNet [online]. 18.9.2013 [cit. 2014-04-29]. Dostupné z: http://www.zdnet.com/rdbms-vs-nosql-how-do-you-pick-7000020803/

[14] FRANCIA, Steve. *REST vs SOAP, the difference between soap and rest.* Hacking Management [online]. 15.1.2010 [cit. 2014-04-29]. Dostupné z: http://spf13.com/post/soap-vs-rest

[15] HARVATH, Mike. *Finding Your Company's Growth-Rate Sweet Spot.* Redmond Channel Partner [online]. 12.8.2013 [cit. 2014-04-29]. Dostupné z: http://rcpmag.com/articles/2013/07/01/grow-just-right.aspx

[16] *Market Growth.* Frost & Sullivan [online]. [cit. 2014-04-29]. Dostupné z: http://www.frost.com/prod/servlet/mcon-mktmeasures-mkt-growth.pag

[17] *How is Actual Sales Growth Rate calculated?* Financial Tools [online]. 26.2.2010 [cit. 2014-04-29]. Dostupné z: http://www.financialtools.com/knowledgebase/how-is-actual-sales-growth-rate-calculated/

[18] WEBER, John a Utpal DHOLAKIA. *Planning Market Share Growth in Mature Business Markets.* In: Industrial Marketing Management [online]. 1998 [cit. 2014-04-29]. Dostupné z: http://goo.gl/sxdmbb